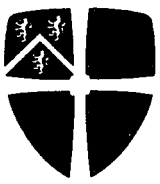
Robust data analysis for factorial experimental designs: Improved methods and software

Majid Sarmad

A Thesis presented for the degree of
Doctor of Philosophy



Statistics
Department of Mathematical Sciences
University of Durham
England

November, 2006

Dedicated to

My loving father who was the first to encourage me to study hard. The rest of my academic life will be dedicated to his memory.

Moreover, it is dedicated to my respectful mother whose magnanimity cannot be expressed in words alone.

My beloved wife and the kids.

Robust data analysis for factorial experimental designs: Improved methods and software

Majid Sarmad

Submitted for the degree of Doctor of Philosophy
November 2006

Abstract

Factorial experimental designs are a large family of experimental designs. Robust statistics has been a subject of considerable research in recent decades. Therefore, robust analysis of factorial designs is applicable to many real problems. Seheult and Tukey (2001) suggested a method of robust analysis of variance for a full factorial design without replication. Their method is generalised for many other factorial designs without the restriction of one observation in each cell. Furthermore, a new algorithm to decompose data from a factorial design is introduced and programmed in the statistical computer package R. The whole procedure of robust data analysis is also programmed in R and it is intended to submit the library to the repository of R software, CRAN. In the procedure of robust data analysis, a cut-off value is needed to detect possible outliers. A set of optimum cut-off values for univariate data and some dimensions of two-way designs (complete and incomplete) has also been provided using an improved design of simulation study.

Declaration

The work in this thesis is based on research carried out at the Statistics Group, the Department of Mathematical Sciences, England. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Acknowledgements

I would like to express my gratitude to Dr. Peter S. Craig for his support and scientific supervision during my study in Durham. I will keep in my mind his sympathy, knowledge and experience in computational statistics. I hope I can learn more from him even after this degree. I have appreciated the help and kindness of Dr. Allan H. Seheult from the first weeks of my study until now. I would like to acknowledge the friendly environment of the Department of Mathematics where I made many friends, too many to mention individually here. I dedicate my degree to Ferdowsi University of Mashhad which was my academic birth place and am very thankful of the Ministry of Science, Research and Technology in Iran who supported my studies. Certainly, I need to appreciate my wife, Soheyla for her patience during my study, Mahdi, Mohammad Saeed and Fatemah for their sweet smells and who will hopefully be the new researchers of tomorrow.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robustness in statistics has been considered in recent decades. It needs to be studied due to departures from the usual modelling assumptions in real data. From the following quotes, it may be understood where robust statistics lies and how important it is to pay more attention to robust statistical methods.

"The study of robust statistics is useful for anyone who handles random data. Applications can be found in statistics, economics, engineering, information technology, psychology, and in the biological, environmental, geological, medical, physical and social sciences."(Olive 2006)

"There are many classical statistical procedures such as least squares estimation for multiple linear regression and the t-interval for the population mean μ. A given classical procedure should perform reasonably well if certain assumptions hold, but may be unreliable if one or more of these assumptions are violated. A robust analog of a given classical procedure should also work well when these assumptions hold, but the robust procedure is generally tailored to also give useful results when a single, specific assumption is relaxed."(Olive 2006)

"A statistical method is said to be robust if its behavior is relatively insensitive to slight departure from the assumptions that justify that method."(Kotz et al. 1988; page 176)

"By a resistant technique will be meant one whose results are at most mildly affected by observations which do not conform to the general pattern of the data."(Besag 1981)

"In recent years, considerable attention has been paid to the development of resistant techniques for data analysis, primarily under the influence of J.W.Tukey and his coworkers."(Besag 1981)

"I think it is an important area that is used a lot less than it ought be."(Ripley 2004)

1.1 Robustness in statistics

"In the statistical literature the word *robust* is synonymous with *good.*"(Olive 2006) Hampel (2001) states that almost everybody nowadays believes in the "dogma of Normality" as an assumption for the error distribution to find the "best estimate" of an unknown "true value". He also mentions that "Bessel (1818), Newcomb (1886), Jeffreys (1939) and others, showed that typical error distributions of high-quality data are slightly but clearly longer-tailed than the Normal." He adds "... obviously real data have different accuracy, as modelled by Newcomb (1886)." It seems that it is risky to trust the assumptions in an statistical analysis and an alternative is to switch to robust statistics.

Huber (1972) also believed it was dogmatic to assume Normality for the errors: "It seems to me that this kind of discussion borders on dogmatism; a more rational action would have been to look at actual error distributions in large samples, to check whether they were compatible with a Normal and, if not, to develop a different theory of estimation."

"Gross errors often show themselves as outliers, but not all outliers are gross errors. Some outliers are genuine and may be the most important observations of the sample. For example, if a geodetic point seems suddenly to be in a different position, it may mean a gross error of some sort, or it may mean a shift of the underground, and some redundancy (or experience) is needed to distinguish these possibilities."(Hampel 2001)

Considering the above realities, there are some issues to address in order to deal with the problem. One issue is to determine the outliers, if any, and then find a way to deal with them. Recognising suspected outliers has been addressed in the litera-

ture by such as Barnett and Lewis (1994), Iglewicz and Hoaglin (1993), Rousseenuw and Leroy (1987), Bhar and Gupta (2001), Rosner (1975; 1983), Stefensky (1972), Shiffler (1988), Daniel (1960), Al-Madfai (1994), Johnson (1989) and Seheult and Tukey (2001) for some different statistical types of data collection; however, many types of data collection have not been covered. Many of the existing methods to detect outliers need a cut-off, or in other words a criterion which plays a key role. Fixing an optimum cut-off value usually takes a separate study.

The next issue that arises is how to manage detected outliers. "A common reaction to this danger is 'rejection of outliers' ... ." (Hampel 2001) Quoting Hampel (1985), he also says that, even using a good rule to detect the outliers, rejecting them, and then doing least squares for the remaining data, typically loses 10-20 percent efficiency compared with a better robust method. It is also worth quoting Ripley (2004) mentioning that screening data and removing the outliers is not sufficient:

"1. Users, even expert statisticians, do not always screen the data.

2. The sharp decision to keep or reject an observation is wasteful. We can do better by down-weighting dubious observations than by rejecting them, although we may wish to reject completely wrong observations.

3. It can be difficult or even impossible to spot outliers in multivariate or highly structured data.

4. Rejecting outliers affects the distribution theory, which ought to be adjusted. In particular, variances will be underestimated from the 'cleaned' data."

Although rejecting outliers is the simplest way, there are a few alternatives. Tukey suggested substituting each outlier by the next closest number to it. He named it Winsorizing in honour of Charlie Winsor (see Huber 2002). Huber (2002) notes that "Interestingly, and rather counter-intuitively, it turned out a few years later that trimming does exactly what Winsorizing was supposed to do but, on the other hand, that the standard error calculated from the Winsorized sample asymptotically gives the correct value for the standard error of the trimmed mean (see Huber 1981; pages 58–59)." Later Seheult and Tukey (2001) suggested half-Winsorizing which is substituting the outlier by the half of the next closest number

to the outlier. Either trimming or modification of an outlier should be done after being in almost no doubt that the outlier is the result of a gross error.

Another approach is to find the breakdown point of the statistic which is used. "Some sources use the term breakdown bound instead of breakdown point ... . The breakdown point of an estimator is the largest proportion of the data that can be replaced by arbitrary values without causing the estimated value to become infinite." (Iglewicz and Hoaglin 1993; page 11) It might be better to consider the breakdown point before outlier detection as the outlier may have some useful information and need to be kept in its original form. Again from Hampel (2001):"The breakdown point ... is often the first and most important number to be looked at before going into the details of local robustness properties." Using a statistic with a high breakdown point is a good way to do a resistant statistical analysis. "The sample mean y can be upset completely by a single outlier; if any data value $y_i \to \infty$, then $y \to \infty$. This contrasts with the sample median, which is little affected by moving any single value to ∞. We say that the median is resistant to gross errors whereas the mean is not. In fact the median will tolerate up to 50% gross errors before it can be made arbitrarily large; we say its breakdown point is 50% whereas that for the mean is 0%." (Ripley 2004)

The fourth approach is to do a robust analysis such as using M-estimators instead of least squares method or median polish in an experimental design.

"It is convenient to divide methods of robustification into four categories:
(1) trimming, Winsorization, and other methods based on other statistics,
(2) M-estimation and minimum distance estimation,
(3) rank statistics, and
(4) outlier tests and diagnostics." (Kotz et al. 1988; page 177)

Finally, a completely different approach is to use a robust design which means a design for which the analysis is resistant in presence of outliers. However, there is only a tiny literature on the production and use of these robust designs while extensive studies have been done on robust designs against missing observations. (Bhar and Gupta 2001; page 339)

1.2 Outliers or Extreme values

Data analysis can be affected by outliers in any type of data collection. Ignoring them, the analysis might be misled to a wrong or far from correct conclusion. "The concept of an outlier has fascinated experimentalists since the earliest attempts to interpret data." (Barnett and Lewis 1994) "That observation is suspect whose removal greatly simplifies the description of the rest of the data. That observation is also suspect that complicates the description of the error distribution." (Daniel 1960) Two other terms for "outlier" are "discordant" and "contaminant". (see Kotz et al. 1988; page 177) "Aberrant" has also been seen in some literature. "Outliers occur very frequently in real data, and they often go unnoticed because nowadays much data is processed by computers, without careful inspection or screening." (Rousseenuw and Leroy 1987)
They can be a result of any of the following reasons: (see Barnett and Lewis 1994, Hawkins 1980)

- A mistake when recording the data - Measurement error
- A mistype in data entry - Measurement error
- Collecting data from a long tail distribution - Inherent variability
- Wrong experiment to collect the sample - Execution error or biased sampling

Regardless of what does cause one or more outliers, they will detract from the results.

"It is clear that a single outlier - if located sufficiently far away - can completely spoil a least squares analysis."(Hampel 2001)

"Outliers are sample values that cause surprise in relation to the majority of the sample. This is not a pejorative term; outliers may be correct, but they should always be checked for transcription errors. They can play havoc with standard statistical methods, and many robust and resistant methods have been developed since 1960 to be less sensitive to outliers."(Ripley 2004)

The following examples show how an outlier can affect a statistical analysis.

1.2.1 Example 1: A univariate case

Suppose, as a hypothetical sample, we have

$$3, 5, 4, 6, 3, 20$$

It seems obvious that for some reason the value 20 is a mistake. A 95% confidence interval for the population mean, assuming the data are from a Normal distribution, from the sample is

$$(-0.04561, 13.7123)$$

which includes zero. The researcher may draw a conclusion of not rejecting the null hypothesis that the population mean is zero.
Let's assume the correct value for the last observation is 2. Then the 95% mean confidence interval for the correct hypothetical sample

$$3, 5, 4, 6, 3, 2$$

is

$$(2.2886, 5.3781)$$

which does not include zero and leads to rejection of the same null hypothesis.

1.2.2 Example 2: A case of experimental design

In experimental designs, the existence of only one outlier may change the result of analysis of variance (ANOVA). A hypothetical example of a 5 by 4 two-way factorial experimental design (without replication) will be followed by its ANOVA table as given by R.

23	54	52	35
61	23	53	45
29	20	25	38
36	29	32	31
38	38	24	30

```
> summary(aov(y~fr+fc,data=ex.hy))
            Df  Sum Sq Mean Sq F value Pr(>F)
fr           4  829.20  207.30  1.4426 0.2796
fc           3   67.60   22.53  0.1568 0.9233
Residuals   12 1724.40  143.70
```

Substitution of just one datum with a supposed outlier, makes a lot of difference to the Sum of Squares' column and hence to the final results:

123	54	52	35
61	23	53	45
29	20	25	38
36	29	32	31
38	38	24	30

```
> summary(aov(y~fr+fc,data=ex.hy.out))
            Df Sum Sq Mean Sq F value  Pr(>F)
fr           4 3869.2   967.3  3.0511 0.05978 .
fc           3 1887.6   629.2  1.9846 0.17013
Residuals   12 3804.4   317.0
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

Assuming the significance level to be 0.1, the second ANOVA table indicates a significant difference between the levels of the row factor while it does not occur in the first ANOVA table.

Both examples can easily happen by mistyping and when the computations are being done by computer, it could be hidden from the researcher's eye!

One may find many other examples in other statistical data such as regression or time series. We will try to find how to deal with outliers in the univariate case as a preface to studying experimental designs.

1.3 Methods of outlier detection

There are many existing methods to detect possible outliers in a univariate data set. Most of them may be generalised to apply for some types of experimental designs.

However, there are also a few specially constructed methods for data which are not univariate.

1.3.1 Univariate outlier detection

There are many methods to detect the outliers in a univariate data set. Most have been constructed first to detect a single outlier. Some of them have been generalised to detect two outliers or more. There are still some methods with weaknesses which can be improved. Among the methods for outlier detection in a univariate data set, we may name ESD (Extreme Studentised Deviate), generalised ESD, MNR (Maximum Normed Residual), Shapiro-Wilk test, L_r, Z-scores, Modified Z-scores, test based on Kurtusis, Dixon-Type test and the boxplot rule. In this part, a few of them will be introduced.

ESD

Grubbs (1950) has a discussion on several solutions to find one outlier in a sample of size n from a reference distribution of Gaussian. Grubbs (1969) also made a tutorial on how to detect two highest data in a sample or two lowest or one highest and one lowest. Iglewicz and Hoaglin (1993) have compared five methods to detect more than one possible outlier. As a result, Generalised ESD (Extreme Studentised Deviate) was proposed by Rosner (1975) who introduced the procedure to detect more than one outlier and prepared a table for many sample sizes $n \geq 25$ (Rosner 1983).

Z-Scores

This is a popular rule to tag possible outliers in a univariate data set. It is just based on the property of Gaussian distribution that 99.7% of values lie between -3 and 3. The method has an easy computational procedure and so many people like to use it. The problem, revealed by Shiffler (1988), is its inability to detect the outliers in sample size less than 11.

Modified Z-Scores

Iglewicz and Hoaglin (1993) mentioned the weaknesses of using Z-scores to detect outliers. An alternative was using resistant estimators rather than the sample average and the sample standard deviation which both can be affected even by one outlier. Then all Z-scores including non-outliers will be changed. "The sample mean, standard deviation, and range have breakdown points of zero ..." (Iglewicz and Hoaglin 1993; page 11). Replacing the sample mean by the median, which has a breakdown point of about 50%, is one modification of Z-Score. Modified Z-Score has been made by also replacing the standard deviation of the sample by the median of absolute deviations of observations from the median, i.e. $MAD = \text{median}_i\{\mid x_i - \tilde{x} \mid\}$ called MAD where $\tilde{x}$ is sample median. Because, for very large n, $E(MAD) = 0.6745\sigma$, the modified Z-Score introduced by Iglewicz and Hoaglin (1993) uses the scaling constant 0.6745 which however seems inappropriate to use for all sample sizes.

1.3.2 Methods for experimental designs

"In classical and in robust analyses, residuals are important; in exploratory work, residuals are generally of paramount importance." (Besag 1981) By studying the residuals, there are a few methods to recognise the possible outliers in a two-way factorial designs. Usually they have been improved from their first versions and mostly restricted to the full two-way factorial design without replication.

MNR in factorial designs

Daniel (1960) proposed a statistic equivalent to the maximum normed residual for detecting a single bad value in a factorial design with replication (see Stefensky 1971; 1972). Later, Stefensky (1972) generalised the cut-off values given by Daniel (1960).

Least square methods

Iglewicz and Hoaglin (1993; page 67-68) discussed the problems of using the same rules to identify the outliers in multiple regression for a balanced factorial experimental design.

Modified R charts

"Ullman (1989) uses a quality control technique called analysis of ranges to provide a simple tool for identifying outliers in replicated factorial experiments." (Iglewicz and Hoaglin 1993) They (Iglewicz and Hoaglin 1993) also simplified Ullman's approach when the number of cells is at least 12.

Developed Cook's Statistic

Bhar and Gupta (2001) modified Cook's statistic in a way to detect a single outlier in a randomised complete block design. Cook's statistic is usually used for a linear regression. The newly developed statistic for a block design can be calculated for each observation in the design and then using a criterion each will be checked for being an outlier if it corresponds to an influential observation. They defined an influential observation to be one which, when removed from the data, will change the F-ratio in an ANOVA table so that the decision of rejecting or not rejecting the null hypothesis will consequently be changed.

Tukey in factorial designs

It seems that the unpublished report by Fowlkes, McRae, Seheult, and Tukey (1981) is the beginning of their work on detecting and dealing with the outliers in the two-way factorial designs. Seheult (2006) believes the main idea was from Tukey. The method is to polish the table first by a resistant statistic. Tukey (1977) introduced the main idea of polishing. Polishing a two-way table leaves four partitions: the total effect, row effects, column effects and the interaction or residuals. The next step was inspecting the residuals to find and tag the possible outliers. A way to find the outlier has been introduced by Daniel (1960) using the half-normal distribution.

The positive half-normal order statistics are calculated in Tukey's method as a set of reference values. Then by comparing the residuals and the reference values, the magnitude of each residual is revealed. Scaling them by a resistant statistic makes them ready to tag the outliers.

The median is a resistant statistic. In an odd number of data, there is one datum in the middle. In an even number of data, there are two data in the middle and in addition to these two, any value between them can be chosen as the median. Tukey has used the lower value called "lo-median" in his first works while in Seheult and Tukey (2001), they chose one of the lower or the higher value in middle based on a definition in the procedure of polishing. This index function of lower and higher values is called "fibian" in Seheult and Tukey (2001). The situation with most advantage gained from using the fibian is in the two-way table with one factor of two levels.

Similar to other outlier detection methods, what is needed again is a cut-off value at the last stage of tagging. Based on this method, Fowlkes et al. (1981) and Al-Madfai (1994) have done some simulations to find out an optimum cut-off value for a 5 by 4 two-way table. Although, Al-Madfai has used an additional scaling called "SqLm", the results of the two simulation studies are very close.

Present work is a generalised form of Tukey's method

Selecting one of the lo-median or hi-median to be called fibian is not so easy for programming. Instead, "NE-median" introduced by Johnson (1989) has similar properties as fibian but is easier to program because it is a stand-alone function whereas fibian needs to keep track of and use each effect during the decomposition. Like fibian, NE-median is also one of lo-median or hi-median but is simply the one which is closest to zero.

In the present work, there are some improvements and generalisations which have been applied on Tukey's method in outlier detection:

- Truncating the reference distribution to produce outliers in the simulation study

- Adding some random normal numbers to the row and column effects as a generalised form of simulation
- Using NE-median to polish in simulation study
- Performing a unique polish
- Using an improved approximation for the reference values given in Seheult and Tukey (2001)
- Applying the simulation for many more sets of number of levels in the two-way tables
- Finding the accuracy of optimum cut-off values
- An independent simulation to find the scaling item
- A very much bigger simulation

1.4 Robust designs

By robust designs we mean "designs are constructed that guard against particular shortcomings." (Kotz et al. 1988) "Robustness of designs against missing observations has been studied extensively in the literature. Only a little work on robustness against the presence of outliers is found in the literature." (Bhar and Gupta 2001) Box and Draper (1975) were the first to study robust response surface designs. Kotz et al. (1988) say that Box and Draper (1975) wanted to obtain designs that made the analysis insensitive to outliers. Bhar and Gupta (2001) introduced a few robust designs which are robust against the presence of a single outlier.

1.5 Robust data analysis

"The classical methods – means, variances and least squares – are unsafe. Sometimes they are good, sometimes they are not. ... It is perfectly proper to use both classical and robust/resistant methods routinely, and only worry when they differ enough to

matter. But when they differ, you should think hard. ... What you need is a reasonably self-consistent set of procedures that are reasonably easy to use and reasonably easy to describe." (Tukey 1979)

1.5.1 L1

Burns (1988) showed for some given designs, L1 has not a unique solution. Based on Burns's report, L1 has a unique answer when for an oddly-replicated table (all cells have odd replicates - empty cells are not accepted), and the number of levels for the factor with fewer levels is odd. He adds that it is impossible to have a unique L1 solution when the number of replicates in all cells is even and empty cells are allowed. "It is easily seen that the sum of the absolute value of residuals can never increase at any stage of median polish and, in practice, an L_1 solution is usually attained." (Besag 1981)

1.5.2 Seheult-Tukey

Seheult and Tukey (2001) have completed some previous work by developing an applicable robust ANOVA. The procedure goes back to the 1980s in Princeton University. The first important part of the procedure is tagging outliers if any exist. Decomposition by a resistant statistic and then applying Tukey's method using a fixed cut-off value recognises as many outliers as might exist in the data. A three-way factorial design without replication is discussed in Seheult and Tukey (2001) for the procedure of robust ANOVA. The rest of the procedure after tagging the outliers is to substitute them by half-Winsorized values and then down-sweeping the mean squares in the ANOVA table based on Paul's rule of two which has also been addressed in Hoaglin, Mosteller, and Tukey (1991; chapter 11).

1.6 A new library in R

Statistical analyses are being done by statistical software these days. Statistical packages are full of variations of classical methods to analyse data collected in different patterns for different purposes. It has been said earlier that nobody should

ignore possible outliers. A lack of robust statistical computations is obvious in many statistical software although the theoretical basis exists. "Robust statistical methods are designed to work well when classical assumptions, typically normality and/or the lack of outliers, are violated. Almost everyone agrees on the value of robust statistical procedures. Nonetheless, after more than 40 years and thousands of papers, few robust methods were available in standard statistical software packages until very recently." (Stromberg 2004)

"One reason that contributes to the limited use of Robust Statistics is the heavy computational cost of many of these techniques. The lack of easy to use and well documented computer code does not help either. In the last few years the consolidation of the R-project as a widely available, powerful and versatile computer program for statistical analysis has resulted in many people simultaneously developing and publishing R code that implements Robust Statistics techniques." (Filzmoser 2006)

"Robust Statistics deals with a very real problem in statistical applications: the effect of violations to the model used to analyse the data. The last 40 years have been tremendous advancements in the theory of Robust Statistics, but unfortunately many of these procedures are not widely used in practice yet." (Filzmoser 2006)

A set of functions in R have been provided to do a robust ANOVA for an experimental design. The method used in this package is based on Seheult and Tukey (2001) and on the new method of decomposition proposed in chapter 4. Many types of factorial experimental designs have been checked by this library. By the method, a given data set from a designed experiment should be first decomposed using a sweep function. The library is able to accept any built-in or user-defined sweep function to decompose but the default sweep function is NE.median which was introduced earlier. The algorithm to decompose a design is new and has the capability to work on most factorial designs. Unlike the approach using sweep operators for least squares of Wilkinson (1970), the new algorithm is a hierarchical algorithm. Incidence matrix and terms levels in the model are the main keys in the algorithm of decomposition. It is intended to submit the library to CRAN (the Comprehensive R Archive Network).

1.7 Objectives

The overall objective in this work is to provide a set of functions in R to do a robust data analysis for the family of factorial experimental designs.

The overall objective may be subdivided into

- Finding a method of decomposition which generalises median polish and is applicable to a wide range of linear models for data from factorial designs.
- Improving and generalising previous work (including simulation studies) on outlier detection for median polish in two-way tables.
- Generalising the robust ANOVA procedure of Seheult and Tukey (2001) to a wide range of factorial designs and models.

Chapter 2

Detecting outliers in univariate data

In this chapter two old methods for detecting outlier(s) in univariate data will be explained via a common example. Then the method introduced by Fowlkes et al. will be discussed. A set of optimum cut-off values for the third method is obtained by a simulation study. Confidence intervals for optimum cut-offs are discussed in section 2.6. A simulation study comparing the new method and one of the old ones shows the new method to be more efficient. Some properties of the Slash distribution and a table of proportions of detected outliers in Slash and Gaussian samples are also a part of this chapter.

2.1 Introduction

Three methods to distinguish outliers in a univariate data set will be explained via an example. The first method is more popular than the second one despite its weakness. The third method is based on some work of J.W. Tukey. In each method, a cut-off value to detect the outliers is necessary. The usual cut-off value for the first method comes from the property of the standard Gaussian distribution that 99.7% of data lie between -3 and 3. In the second method, we use the cut-off value proposed by Iglewicz and Hoaglin (1993) as a result of a simulation study. For the last method, there is no previous work to find a cut-off value to detect possible

outliers in a univariate data set. It is the first time that optimum cut-off values for a wide range of sample sizes are being found by a simulation study. The functions to do the simulations are written in R (see Section 2.10 and Appendix B). To generate samples having outliers, the Slash distribution has been used. The details of this distribution will be given later.

Suppose that a single variable has been measured in a randomly selected sample from a population. In the next sections, three methods to distinguish any possible outlier(s) will be described. The key concepts in the methods will be illustrated using the following simple data set:

10, -20, 1, 18, 2, 3, -5, -6, 7, 2, 6, 5, 12, -1, -11, -7, 28, 5, 7, 2

Among the values of the hypothetical sample, it seems likely that -20 and 28 would be considered to be outliers caused, for instance, by data errors or coming from another distribution. We shall now see how the following three methods try to find outliers.

2.2 Method 1: Z-Scores

Using Z-scores is a simple way to try to detect the outliers. Z-scores may be calculated for a set of observations $x_1, x_2, \ldots, x_n$ as follows:

$$z_i = \frac{x_i - \bar{x}}{s}, \text{ where } s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

This method is based on the well-known property of the Normal distribution, i.e:

$$X : N(\mu, \sigma^2) \rightarrow Z = \frac{X - \mu}{\sigma} : N(0, 1)$$

Hence, any Z-score more than 3 in absolute value might be labelled as an outlier. For our given data set, Z-scores are calculated in table 2.1.

It can be seen that no observation with this method is tagged as an outlier. What is happening here is that even a single large $|x_i - \bar{x}|$ causes s to be large too and then it makes an upper bound for z_i. Shiffler (1988) showed that the maximum possible Z-score depends on n (see table 2.2).

Table 2.1: Outlier detection using Z-scores

i	x_i	z_i
1	10	0.6883
2	-20	-2.2199
3	1	-0.1842
4	18	1.4638
5	2	-0.0872
6	3	0.0097
7	-5	-0.7658
8	-6	-0.8628
9	7	0.3975
10	2	-0.0872
11	6	0.3005
12	5	0.2036
13	12	0.8822
14	-1	-0.3781
15	-11	-1.3475
16	-7	-0.9597
17	28	2.4332
18	5	0.2036
19	7	0.3975
20	2	-0.0872

Table 2.2: Maximum possible Z-score depending on sample size n (Shiffler 1988)

n	$Z_{max} = Z_{(n)} = \frac{n-1}{\sqrt{n}}$
3	1.155
4	1.500
5	1.789
10	2.846
11	3.015
17	3.881
18	4.007

Thus, for any sample of size less than 11, no observation could be found as an outlier in this method using cut-off 3 even if you have one or more. For $n > 11$, this method still seems unreliable.

2.3 Method 2: Modified Z-Scores

Iglewicz and Hoaglin (1993) introduced a modified Z-score to detect outliers. This method is based on resistant estimators rather than $\bar{x}$ and s. The modified Z-score is calculated as

$$M_i = \frac{x_i - \tilde{x}}{\frac{MAD}{0.6745}}, \text{ where } \tilde{x} \text{ is the sample median and } MAD = \text{median}_i\{|x_i - \tilde{x}|\}$$

Iglewicz and Hoaglin (1993) say $E(MAD) = 0.6745\sigma$ which is the basis for the constant 0.6745 in M_i. However, the constant introduced by them is for very large n which is not applicable in practice. Running a simple simulation study shows the constant varies for different sample size n.

Accepting this constant for now, table 2.3 shows the procedure and results for the example. Any observation with $|M_i| > D$ is tagged as outlier. Based on a simulation study, they suggested that the cut-off value $D = 3.5$. Therefore, the observation $x_i = 28$ would be suspected as an outlier with this method.

Table 2.3: Outlier detection using Modified Z-scores

i	Data x_i	Ordered x_i	Ordered $\lvert x_i - \tilde{x}\rvert$	$M_i = \frac{0.6745(x_i - \tilde{x})}{median_i\{\lvert x_i - \tilde{x}\rvert\}}$
1	10	-20	0.5	1.1242
2	-20	-11	0.5	-3.3725
3	1	-7	0.5	-0.2248
4	18	-6	0.5	2.3233
5	2	-5	1.5	-0.0749
6	3	-1	2.5	0.0749
7	-5	1	2.5	-1.1242
8	-6	2	3.5	-1.2741
9	7	2	3.5	0.6745
10	2	2	4.5	-0.0749
11	6	3	4.5	0.5246
12	5	5	7.5	0.3747
13	12	5	7.5	1.4239
14	-1	6	8.5	-0.5246
15	-11	7	9.5	-2.0235
16	-7	7	9.5	-1.4239
17	28	10	13.5	3.8222
18	5	12	15.5	0.3747
19	7	18	22.5	0.6745
20	2	28	25.5	-0.0749

2.4 Method 3: Tukey

In this section, finding possible outliers in the example data will be done using the method from Fowlkes, McRae, Seheult, and Tukey (1981). Seheult and Tukey (2001) generalised the method for a set of data collected by a factorial experimental design. The case of a two-way factorial design will be discussed in section 3.2.
To find possible outliers in a univariate sample, it is necessary to construct a table like 2.4. Before that, there are some keys in the table which need to be explained:

- The lower of two data in the middle of an even number of data set is called the *lo-median*. The upper one is called the *hi-median* and the *mid-median* is the average of these two values. Note that for the odd number of values, there is just one median.

- If the number of non-zero residuals is less than the degrees of freedom (in univariate case, it is usually the sample size minus one), only one more than the number of non-zero residuals will be inspected (18 observations in this example). If only one non-zero residual remains, the datum related to the residual would be considered as an unique outlier.

- Using Gaussian as a reference distribution, extreme values are not expected to exist. Outliers may be distinguished by comparing the absolute values of residuals with what would be expected for order statistics from the positive half of Gaussian distribution. The *reference values* in column (4) in table 2.4 are a very good approximation to the expected order statistics for a sample from the positive half of standard Gaussian distribution. There is a short discussion in section 3.5 of how to approximate $E[x_{(i)}]$ where $x_{(i)}$ is i-th order statistic when the sample size is n. Hoaglin, Mosteller, and Tukey (1991; page 187) say a good approximation for $E(x_{(i)})$ is

$$\Phi^{-1}\left(1 - \frac{3i-1}{6n+2}\right)$$

where Φ^{-1} is the inverse Gaussian distribution function and $i = 1$ gives the largest reference values.

In this example, ignoring two zero observations, putting $n = 18$ in the above formula gives the needed reference values.

The ratios of absolute values of residuals to reference values in column (5) should reveal the magnitude of the residuals.

- The lo-median of the ratios is again has been chosen to standardise the magnitudes. It might be called a scaling.

Now a cut-off value has to be chosen to determine which observation should be considered as outlier. Suppose that the cut-off value is to be 1.41. The investigation is started from the bottom of the last column corresponding to the largest absolute residual. Since last scaled ratio is less than the supposed cut-off value (1.41) we would not tag the related observation (28) as possible outlier. Although the next from the bottom is greater than the supposed cut-off value, we do not tag that one as an outlier either, because the last one was not tagged. In fact, no observation will be tagged as an outlier by this method when the cut-off value is 1.41. However, if the cut-off is equal to 1.4, the two last scaled ratios are both greater than 1.4 and the corresponding observations would be tagged as outliers. As mentioned earlier, there is no previous study to find an optimum choice of cut-off for this method. The next section is about a procedure to choose the optimum cut-off, say γ^* when γ denotes any possible cut-off, which is applicable for any outlier detection method which involves a choice of cut-off.

2.5 Optimum cut-off value

To arrive at a sensible choice of cut-off value, γ^*, we choose the value which is the minimax solution for a particular decision problem. The decision problem is designed to choose a value which is a good compromise between what one should do when there are no outliers and what one should do when there are many outliers. Note that we allow the value of γ^* to depend on the sample size n.

For a univariate sample, $x_1, \ldots, x_n$ from a distribution with mean μ, we measure the "badness" of a particular value of γ by $B(\gamma) = (\overline{x}_\gamma - \mu)^2$ where $\overline{x}_\gamma$ is the average

Table 2.4: Outlier detection using Tukey's method

(1)	(2)	(3)	(4)	(5)	(6)
Original data	Data - lo-median (Residuals)	Sorted absolute	Reference values	Ratios=$\frac{(3)}{(4)}$	Scaled ratios $=\frac{(5)}{(5)\text{s' lo-median}}$
10	8	0 (2)	-	-	-
-20	-22	0 (2)	-	-	-
1	-1	0 (2)	0.0456	0	0
18	16	1 (1)	0.1142	8.7566	0.9896
2	0	1 (3)	0.1833	5.4555	0.6165
3	1	3 (5)	0.2533	11.8437	1.3385
-5	-7	3 (-1)	0.3246	9.2421	1.0445
-6	-8	3 (5)	0.3976	7.5453	0.8527
7	5	4 (6)	0.4728	8.4602	0.9561
2	0	5 (7)	0.5507	9.0794	1.0261
6	4	5 (7)	0.6322	7.9089	0.8938
5	3	7 (-5)	0.7180	9.7493	1.1018
12	10	8 (10)	0.8096	9.8814	1.1167
-1	-3	8 (-6)	0.9085	8.8057	0.9951
-11	-13	9 (-7)	1.0171	8.8487	1
-7	-9	10 (12)	1.1394	8.7765	0.9918
28	26	13 (-11)	1.2816	10.1436	1.1463
5	3	16 (18)	1.4558	10.9905	1.2421
7	5	22 (-20)	1.6906	13.0131	1.4706
2	0	26 (28)	2.0928	12.4235	1.4040

of the remaining data having applied the procedure to remove outliers using the specified cut-off value γ. This measure is based on the presumption that the goal of collecting the data is to estimate the population mean and that we use the sample mean (omitting outliers) as the natural estimator.

Our measure of the adverse consequences (across all samples) of a particular choice of γ for a particular population distribution is then $b(\gamma) = \mathrm{E}[\,B(\gamma)\,]$ where the expectation is taken with respect to samples of size n from the distribution. The measure is location-invariant, i.e. it is not changed by changing the mean of the distribution.

For any location-family of distributions, there is an optimal choice of γ which minimises $b(\gamma)$ for that family. The problem for the data analyst is that the family and consequently $b(\gamma)$ and its behaviour is not known to him/her. Hence we seek a compromise choice of γ. To make comparisons between the consequences for different families of distributions, we use the "scaled badness" $b_{\mathrm{sc}}(\gamma) = b(\gamma)/b_m$ where $b_m = \min_\gamma b(\gamma)$. We denote by γ_m the value for which $b(\gamma_m) = b_m$.

There are two key reasons for using the scaled badness measure. First, $b(\gamma)$ is essentially a measure of inefficiency of estimation of the sample mean and so the scaled badness $b_{\mathrm{sc}}(\gamma)$ is effectively a measure of relative inefficiency compared to the best possible for that family. Secondly, the scaled badness is the same for all members of a location-scale family of distributions which corresponds well to the fact that our procedure for determining outliers is location and scale-invariant.

To arrive at the compromise choice of cut-off, we consider two extreme location-scale families of distribution: the Gaussian and Slash families[1], two of Tukey's so-called three corners[2](Yatracos 1991). The former may be considered to generate no outliers and the latter are distributions which generate many outliers. We then find the cut-off which is the minimax solution for the decision problem where the loss function, measuring the adverse consequences of using a particular cut-off for a

[1]Some properties of Slash distribution will be discussed in section 2.5.1

[2]The third corner is one-wild distribution which is a sample of 95% of standard Normal and 5% from $N(0, 100)$. This corner has not been used in the present work as the initial aim was to compare some results with Fowlkes and Al-Madfai who have not used it.

particular location-scale family, is the scaled badness measure. Then, the optimal cut-off is the value γ^* which minimises the function $\max(b_{sc}^G(\gamma), b_{sc}^S(\gamma))$ where $b_{sc}^G(.)$ and $b_{sc}^S(.)$ are respectively the scaled badness functions for the Gaussian and Slash families of distributions.

2.5.1 Slash Distribution

The Slash distribution has been chosen by previous authors as a distribution suitable for this type of simulation study because it is similar to the Normal distribution in the middle but generates many extreme values.

Probability density function of Slash distribution

Let $S = \frac{X}{Y}$, where $X \sim N(0,1)$ and $Y \sim U(0,1)$ are independent. Then S is a random variable with Slash distribution. Let $W = Y$. Then, the Jacobian of this transformation is

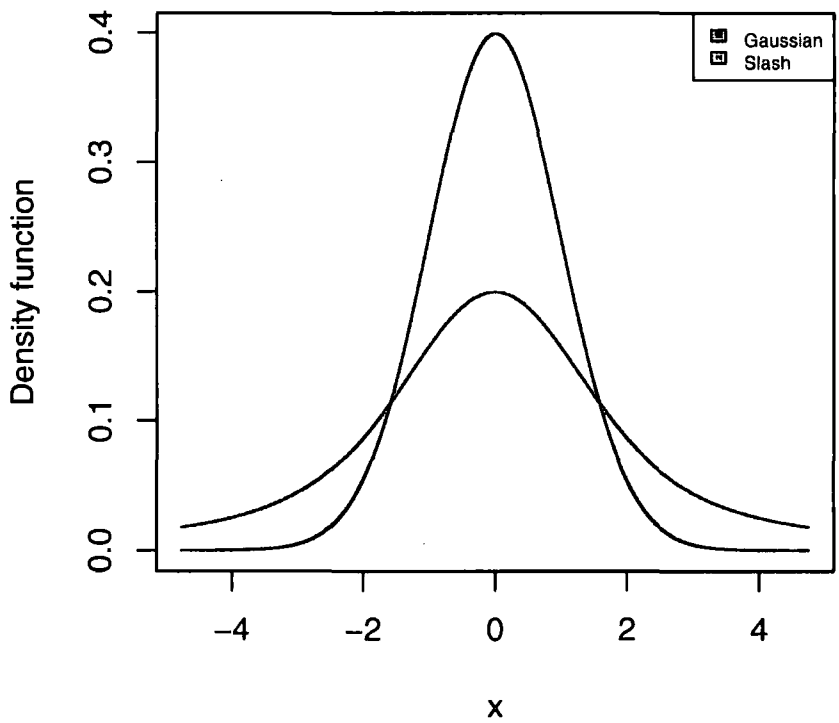
$$J = \frac{\partial(s,w)}{\partial(x,y)} = \begin{vmatrix} \frac{\partial s}{\partial x} & \frac{\partial w}{\partial x} \\ \frac{\partial s}{\partial y} & \frac{\partial w}{\partial y} \end{vmatrix} = \begin{vmatrix} \frac{1}{y} & 0 \\ \frac{-x}{y^2} & 1 \end{vmatrix} = \frac{1}{y} \Rightarrow J^{-1} = y$$

and the joint density function of s and w is given by $g(s,w) = f(x,y)J^{-1} = \phi(x)y = \phi(sw)w = \frac{w}{\sqrt{2\pi}} e^{-\frac{s^2w^2}{2}}$ where $0 \leq w \leq 1$, $-\infty < s < +\infty$. Finally, the density function of s is

$$\begin{aligned} f_S(s) = \int_0^1 g(s,w)dw &= \frac{1}{\sqrt{2\pi}} \int_0^1 we^{-\frac{s^2w^2}{2}} dw \\ &= \frac{1}{\sqrt{2\pi}} \frac{-1}{s^2}(e^{-\frac{s^2}{2}} - 1) \\ &= \frac{1}{s^2}(-\frac{1}{\sqrt{2\pi}} e^{-\frac{-s^2}{2}} + \frac{1}{\sqrt{2\pi}}) \\ &= \frac{1}{s^2}(\phi(0) - \phi(s)) \end{aligned}$$

Figure 2.1 shows the Slash probability density function compared to the standard Normal distribution.

Figure 2.1: Slash density function; Similar to Normal in the middle, long tails



Distribution function of Slash

First

$$\begin{aligned} P(S \leq s|U=u) &= P(\frac{Z}{U} \leq s|U=u) \\ &= P(\frac{Z}{u} \leq s|U=u) \\ &= P(Z \leq su|U=u) \\ &= P(Z \leq su) = \Phi(su). \end{aligned}$$

Now

$$\begin{aligned} F_S(s) = P(S \leq s) &= \int_0^1 P(S \leq s|U=u) f_U(u) du \\ &= \int_0^1 \Phi(su) du \\ &= \frac{1}{s} \int_0^s \Phi(w) dw \\ &= \frac{1}{s} \left(w\Phi(w) \mid_0^s - \int_0^s w\phi(w) dw \right) \\ &= \frac{1}{s} \left(s\Phi(s) - \frac{1}{\sqrt{2\pi}} \int_0^{\frac{1}{2}s^2} e^{-y} dy \right) \\ &= \Phi(s) - \frac{1}{s\sqrt{2\pi}} \left(-e^{-y} \mid_0^{\frac{1}{2}s^2} \right) \\ &= \Phi(s) - \frac{1}{s} (\phi(0) - \phi(s)) \end{aligned}$$

2.6 Simulation study - Basic notations

In practice, $b_{sc}^S(\gamma)$ is an increasing function of γ (except for a short initial section near 1 and $b_{sc}^G(\gamma)$ is a decreasing function of γ and so γ^* may be found by solving the equation $b_{sc}^G(\gamma^*) = b_{sc}^S(\gamma^*)$. Writing $\delta(\gamma) = b_{sc}^G(\gamma) - b_{sc}^S(\gamma)$, γ^* must satisfy $\delta(\gamma^*) = 0$.

None of $b_{sc}^G(\gamma)$, $b_{sc}^S(\gamma)$ and $\delta(\gamma)$ are known but they may be estimated for a set of γ's, e.g. $\gamma_1, \gamma_2, \cdots, \gamma_k$, using a simulation study.

Practically, for a given sample size n, N_1 samples of size n from Gaussian and N_2 samples of size n from Slash may be generated. Although, to reduce the variance of the difference between the average of badnesses in two sets of samples i.e. Gaussian and Slash samples, it would be better to generate the Gaussian samples from the

Slash samples using the following conversion:

$$z = \Phi^{-1}\left(F_S(s)\right).$$

Obviously $z \sim N(0,1)$.

Therefore, what we need is to generate firstly N samples from Slash and then N correlated Gaussian samples corresponding to each of Slash samples. By applying the procedure to detect the outliers on each sample using any of γ's in the set, N sets of badnesses for Slash samples and N sets of badnesses for Gaussian samples will be obtained:

$$\begin{array}{cccc} B_1^S(\gamma_1) & B_1^S\gamma_2) & \cdots & B_1^S(\gamma_k) \\ B_2^S(\gamma_1) & B_2^S(\gamma_2) & \cdots & B_2^S(\gamma_k) \\ & & \ddots & \\ B_N^S(\gamma_1) & B_N^S(\gamma_2) & \cdots & B_N^S(\gamma_k). \end{array}$$

and

$$\begin{array}{cccc} B_1^G(\gamma_1) & B_1^G(\gamma_2) & \cdots & B_1^G(\gamma_k) \\ B_2^G(\gamma_1) & B_2^G(\gamma_2) & \cdots & B_2^G(\gamma_k) \\ & & \ddots & \\ B_N^G(\gamma_1) & B_N^G(\gamma_2) & \cdots & B_N^G(\gamma_k) \end{array}$$

Now what we have is k badnesses for each N samples of Slash and Gaussian. Thus we estimate the expected badnesses by

$$B^S(\gamma_i) = \frac{1}{N}\sum_{j=1}^{N} B_j^S(\gamma_i)$$

and

$$B^G(\gamma_i) = \frac{1}{N}\sum_{j=1}^{N} B_j^G(\gamma_i)$$

for $i = 1, 2, \cdots, k$.

To scale the badnesses, we need the minimum of badness for each distribution.

Gaussian samples: b_m^G can be obtained theoretically:

In Gaussian samples, we know that the optimal estimator of the population mean is the sample mean. Therefore, by removing even one observation, this estimator has not been used and the measure of badness will be increased as a result. In the other words, minimum of measure of badness occurs when no observation is

removed. To keep all observations of all Gaussian sample, γ_m^G must be ∞. Keeping all observations leads the badness to be the same as variance of the mean which is $\frac{1}{n}$.

Slash samples: It is not known where exactly γ_m^S is and we may estimate it using the simulation results. Let

$$B_m^{S'} = min(B^S(\gamma_1), B^S(\gamma_2), \cdots, B^S(\gamma_k))$$

which is a negatively biased estimator of b_m^S and then define $\hat{\gamma}_m^S$ to satisfy $B^S(\hat{\gamma}_m^S) = B_m^{S'}$. Now, an independent simulation study just at $\hat{\gamma}_m^S$ (no need for a set of γ's) leads to an unbiased estimator of $b^S(\hat{\gamma}_m^S)$ denoted by B_m^S. It is still a biased estimator (positively this time) for $b_m^S = b^S(\gamma_m^S)$, but the second bias is never great and is usually less than the first one (Craig 2006) (see Appendix A).

At the first stage of the simulation study, using a wider range of γ's, γ^* can be initially estimated from a graph of $\hat{b_{sc}^S}(\gamma_i) = \frac{B^S(\gamma_i)}{B_m^{S'}}$ and $\hat{b_{sc}^G}(\gamma_i) = \frac{B^G(\gamma_i)}{b_m^G}$ for $i = 1, 2, \cdots, k$. We note the initial optimum cut-off value as $\hat{\gamma^*}'$.

The next stage is to estimate more accurately γ^* and its accuracy. Doing another set of simulations with the similar procedure but for a set of γ's near to the initial estimate of γ^* may help. A graph is still applicable to estimate γ^*. However, another way to solve $\delta(\gamma^*) = 0$, which also helps to find an accuracy for γ^*, is to approximate $\delta(\gamma)$ by a line

$\alpha + \beta\gamma$. Now $\delta(\gamma^*) \simeq \alpha + \beta\gamma^* = 0$ leads to the answer

$$\gamma^* = -\frac{\alpha}{\beta}.$$

α and β can be estimated by running a regression of $\hat{\delta}(\gamma_i) = \hat{b_{sc}^S}(\gamma_i) - \hat{b_{sc}^G}(\gamma_i)$ on γ_i.

The benefit of the above solution is to make it possible to find the uncertainty of $\hat{\gamma^*} = -\frac{\hat{\alpha}}{\hat{\beta}}$:

Let $f(\alpha, \beta) = -\frac{\alpha}{\beta}$ which is the quantity we need a confidence interval for. Again based on Taylor's expansion about the point $(\hat{\alpha}, \hat{\beta})$, having $\frac{\partial f}{\partial \alpha} = -\frac{1}{\beta}$ and $\frac{\partial f}{\partial \beta} = \frac{\alpha}{\beta^2}$, $f(\alpha, \beta) \simeq -\frac{\hat{\alpha}}{\hat{\beta}} - \frac{1}{\hat{\beta}}(\alpha - \hat{\alpha}) + \frac{\hat{\alpha}}{\hat{\beta}^2}(\beta - \hat{\beta}) = -\frac{\hat{\alpha}}{\hat{\beta}} - \frac{1}{\hat{\beta}}\alpha + \frac{\hat{\alpha}}{\hat{\beta}^2}\beta$.

Considering $\underline{C}^T = (-\frac{1}{\hat{\beta}}, \frac{\hat{\alpha}}{\hat{\beta}^2})$ and $\underline{\theta}^T = (\alpha, \beta)$, an approximate $(1 - \alpha)\%$ confidence interval for $\underline{C}^T\underline{\theta} = -\frac{1}{\hat{\beta}}\alpha + \frac{\hat{\alpha}}{\hat{\beta}^2}\beta$ is (see Seber 1977; page 108)

$$\underline{C}^T\hat{\underline{\theta}} \pm Z_{\frac{\alpha}{2}}\sqrt{\underline{C}^T Var(\hat{\underline{\theta}})\underline{C}}$$

and so the confidence interval for $f(\alpha, \beta)$ is simply

$$-\frac{\hat{\alpha}}{\hat{\beta}} + \underline{C}^T\underline{\hat{\theta}} \pm Z_{\frac{\alpha}{2}}\sqrt{\underline{C}^T Var(\underline{\hat{\theta}})\underline{C}}. \tag{2.1}$$

We know that $Var(\underline{\hat{\theta}}) = (\underline{X}^T\underline{X})^{-1}\underline{X}^T Var(\underline{\epsilon})\underline{X}(\underline{X}^T\underline{X})^{-1}$ where

$$X = \begin{pmatrix} 1 & \gamma_1 \\ 1 & \gamma_2 \\ \vdots & \vdots \\ 1 & \gamma_k \end{pmatrix}.$$

and $Var(\underline{\epsilon}) = \sigma^2 I$ if the errors of the regression of $\hat{\delta}(\gamma_i)$ on γ_i are independent. Assuming b_m^S is known

$$\hat{\delta}(\gamma_i) = \Delta_i = \frac{B^G(\gamma_i)}{b_m^G} - \frac{B^S(\gamma_i)}{b_m^S}$$

and if we do a separate simulation for each γ_i, the above independence can happen. Thus the confidence interval is obtained by

$$-\frac{\hat{\alpha}}{\hat{\beta}} + C^T\underline{\hat{\theta}} \pm t_{n-2,\frac{\alpha}{2}} S\sqrt{C^T(X^TX)^{-1}C}. \tag{2.2}$$

However, in fact, b_m^S is uncertain and might be estimated by either $B_m^{S'}$ or B_m^S. In either case, there is a dependence for $\hat{\delta}(\gamma_i)$'s and then $Var(\underline{\epsilon}) \neq \sigma^2 I$.

Let us choose B_m^S to estimate b_m^S. Then

$$\hat{\delta}(\gamma_i) = \Delta_i^* = \frac{B^S(\gamma_i)}{B_m^S} - \frac{B^G(\gamma_i)}{b_m^G}$$

The first bit can be written as

$$\begin{aligned}
\frac{B^S(\gamma_i)}{B_m^S} &= \frac{b^S(\gamma_i)+(B^S(\gamma_i)-b^S(\gamma_i))}{b_m^S+(B_m^S-b_m^S)} = \frac{b^S(\gamma_i)(1+\frac{B^S(\gamma_i)-b^S(\gamma_i)}{b^S(\gamma_i)})}{b_m^S(1+\frac{B_m^S-b_m^S}{b_m^S})} \\
&\simeq \frac{b^S(\gamma_i)}{b_m^S}(1+\frac{B^S(\gamma_i)-b^S(\gamma_i)}{b^S(\gamma_i)})(1-\frac{B_m^S-b_m^S}{b_m^S}) \\
&\quad \text{provided } \frac{B_m^S-b_m^S}{b_m^S} << 1 \\
&= \frac{b^S(\gamma_i)}{b_m^S}(1-\frac{B_m^S-b_m^S}{b_m^S}+\frac{B^S(\gamma_i)-b^S(\gamma_i)}{b^S(\gamma_i)}-\frac{B^S(\gamma_i)-b^S(\gamma_i)}{b^S(\gamma_i)}.\frac{B_m^S-b_m^S}{b_m^S}) \\
&\simeq \frac{b^S(\gamma_i)}{b_m^S}(1-\frac{B_m^S-b_m^S}{b_m^S}+\frac{B^S(\gamma_i)-b^S(\gamma_i)}{b^S(\gamma_i)}) \\
&\quad \text{provided } \frac{B^S(\gamma_i)-b^S(\gamma_i)}{b^S(\gamma_i)} << 1 \\
&= \frac{b^S(\gamma_i)}{b_m^S}-\frac{b^S(\gamma_i)(B_m^S-b_m^S)}{(b_m^S)^2}+\frac{b^S(\gamma_i)B^S(\gamma_i)}{b_m^S b^S(\gamma_i)}-\frac{(b^S(\gamma_i))^2}{b_m^S b^S(\gamma_i)} \\
&= \frac{B^S(\gamma_i)}{b_m^S}-\frac{b^S(\gamma_i)}{(b_m^S)^2}(B_m^S-b_m^S)
\end{aligned}$$

Both $\frac{B_m^S-b_m^S}{b_m^S} << 1$ and $\frac{B^S(\gamma_i)-b^S(\gamma_i)}{b^S(\gamma_i)} << 1$ can be verified. The first one is true provided $E[(B_m^S-b_m^S)^2] = Var[B_m^S] + (E[B_m^S]-b_m^S)^2 << (b_m^S)^2$. Thus, it is needed that $SD[B_m^S] << b_m^S$ or, using their estimates, that $SE[B_m^S] << B_m^S$ which can be checked using the simulation results. Also, $E[B_m^S]-b_m^S$ needs to be small. A similar calculation verifies the second.

Now

$$\Delta_i^* = \frac{B^S(\gamma_i)}{b_m^S}-\frac{b^S(\gamma_i)}{(b_m^S)^2}(B_m^S-b_m^S)-\frac{B^G(\gamma_i)}{b_m^G} = \Delta_i - \frac{b^S(\gamma_i)}{(b_m^S)^2}(B_m^S-b_m^S)$$

By noting that $E[B_m^S] \simeq b_m^S$ which is a constant

$$E[\Delta_i^*] = E[\Delta_i]$$

and

$$Var[\Delta_i^*] = Var[\Delta_i] + \frac{(b^S(\gamma_i))^2}{(b_m^S)^4}Var[B_m^S].$$

provided $Cov[\Delta_i, B_m^S] = 0$ which is achieved when calculation of B_m^S comes from a separate simulation.

For any $i \neq j$, Δ_i^* and Δ_j^* are not independent as they both include a random

common term of B_m^S. Then

$$Cov[\Delta_i^*, \Delta_j^*] = Cov[\Delta_i, \Delta_j] - Cov[\Delta_i, \frac{b^S(\gamma_j)}{(b_m^S)^2}(B_m^S - b_m^S)] - Cov[\Delta_j, \frac{b^S(\gamma_i)}{(b_m^S)^2}(B_m^S - b_m^S)] + Cov[\frac{b^S(\gamma_i)}{(b_m^S)^2}(B_m^S - b_m^S), \frac{b^S(\gamma_j)}{(b_m^S)^2}(B_m^S - b_m^S)]$$

and

$$\begin{aligned} Cov[\Delta_i^*, \Delta_j^*] &= Cov[\frac{b^S(\gamma_i)}{(b_m^S)^2}(B_m^S - b_m^S), \frac{b^S(\gamma_j)}{(b_m^S)^2}(B_m^S - b_m^S)] \\ &= \frac{b^S(\gamma_i)}{(b_m^S)^2}\frac{b^S(\gamma_j)}{(b_m^S)^2}Cov[(B_m^S - b_m^S), (B_m^S - b_m^S)] \\ &= \frac{b^S(\gamma_i)}{(b_m^S)^2}\frac{b^S(\gamma_j)}{(b_m^S)^2}Var[B_m^S - b_m^S] \\ &= \frac{b^S(\gamma_i)b^S(\gamma_j)}{(b_m^S)^4}Var[B_m^S]. \end{aligned}$$

Thus, performing a regression of Δ_i^*'s on γ_i's,we have $Var(\underline{\epsilon}) = \sigma^2 I + \underline{\omega}\underline{\omega}^T$ where $\omega_i = \frac{b^S(\gamma_i)}{(b_m^S)^2}SD[B_m^S]$.

Therefore, $Var(\underline{\beta}) = (\underline{X}^T\underline{X})^{-1}\underline{X}^T(\sigma^2 I + \underline{\omega}\underline{\omega}^T)\underline{X}(\underline{X}^T\underline{X})^{-1} = \sigma^2(\underline{X}^T\underline{X})^{-1} + \underline{V}\underline{V}^T$ where $\underline{V} = (\underline{X}^T\underline{X})^{-1}\underline{X}^T\underline{\omega}$. Let $\omega_i = c + d\gamma_i$ for some choice of c and d. Then, $\underline{\omega} = c\underline{1} + d\underline{\gamma}$.

Knowing $\underline{X} = (\underline{1}, \underline{\gamma})$, then $\underline{\omega} = \underline{X}\begin{pmatrix} c \\ d \end{pmatrix}$ and

$$\underline{V} = (\underline{X}^T\underline{X})^{-1}\underline{X}^T\underline{X}\begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix}.$$

Thus,

$$\underline{V}\underline{V}^T = \begin{pmatrix} c^2 & cd \\ cd & d^2 \end{pmatrix}.$$

Note that c and d can be estimated as coefficients of a regression of $\omega_i = \frac{b^S(\gamma_i)}{(b_m^S)^2}SD[B_m^S]$ on γ_i where $\frac{b^S(\gamma_i)}{(b_m^S)^2}SD[B_m^S]$ is estimated by $\frac{B^S(\gamma_i)}{(B_m^S)^2}SE[B_m^S]$.

2.7 Simulation study - The results

The simulation study in this section is to find the optimum cut-off values for Tukey's method. The simulation has been performed for a set of sample sizes

$$n = 4, 5, 6, \cdots, 30, 49, 50.$$

Following the explanation in previous sections, the simulation study should contain three stages:

1. Scanning to find $\hat{\gamma}_m^S$ and making an initial estimate of γ^*
2. Estimation of b_m^S by $B_m^S = B^S(\hat{\gamma}_m^S)$
3. A number of separate simulations to find a more accurate optimum cut-off value ($\hat{\gamma^*}$) and its accuracy

Stages 2 and 3 would be done simultaneously.

As the simulation study for the univariate case was carried out before the full development of this procedure, estimation of b_m^S is by $B_m^{S\prime}$ and so stage 2 is not found in the present univariate simulation study. Instead, stage 1 has been done in substages; one simulation to scan to find $\hat{\gamma}_m$ and another one to find the initial estimate $\hat{\gamma^*}$.

Stage 1. For each given sample size, 10,000,000 samples from a Slash distribution and then by converting from Slash samples, the same number of samples from a correlated Gaussian distribution have been generated. Due to memory restriction and to increase the speed of simulation study, ten million paired simulations have been done in 50 separate simulations containing 200000 samples each.

Then, for a wide range of cut-off values, suspected outliers in each 20,000,000 samples have been detected and removed temporarily, ready to compute the measure of badness. Means of ten million badnesses, separately for Gaussian and Slash, have been calculated for each cut-off value. Table 2.5 shows $\hat{\gamma}_m^S$ and $B_m^{S\prime}$ for a range of sample sizes.

For each n, further simulations were done in the same way but with a different number of samples. This new simulation involved scaling the badnesses by their minimum, obtained at the previous stage. Drawing a graph has been used to intersect the scaled badnesses for Slash and Gaussian. Figure 2.2 shows three different simulations of 1,000,000 samples of size $n = 5$ in the same graph. It can be easily seen that the results are very stable for Gaussian samples and reasonably stable for Slash samples. Figure 2.3 is the average of the three repetitions and is the basis for the optimum cut-off value shown at table 2.6 for $n = 5$.

Table 2.5: Tukey's method: Minimum average badness and corresponding cut-off value according to 10 million simulations for each n

n	$\hat{\gamma}_m^S$	$B_m^{S'}$	n	$\hat{\gamma}_m^S$	$B_m^{S'}$
4	0.99	9.1795	5	0.86	2.1546
6	0.99	1.7110	7	0.99	1.1772
8	0.99	1.0077	9	0.99	0.8191
10	0.99	0.7302	11	1.00	0.6286
12	0.99	0.5769	13	1.00	0.5099
14	1.00	0.4744	15	1.00	0.4296
16	1.00	0.4032	17	1.02	0.3721
18	1.01	0.3513	19	1.04	0.3278
20	1.03	0.3115	21	1.06	0.2931
22	1.05	0.2801	23	1.07	0.2650
24	1.07	0.2542	25	1.09	0.2419
26	1.08	0.2327	27	1.09	0.2223
28	1.09	0.2144	29	1.11	0.2057
30	1.11	0.1987	49	1.18	0.1173
50	1.16	0.1152			

Figure 2.2: Three separate simulations of one million samples of size five

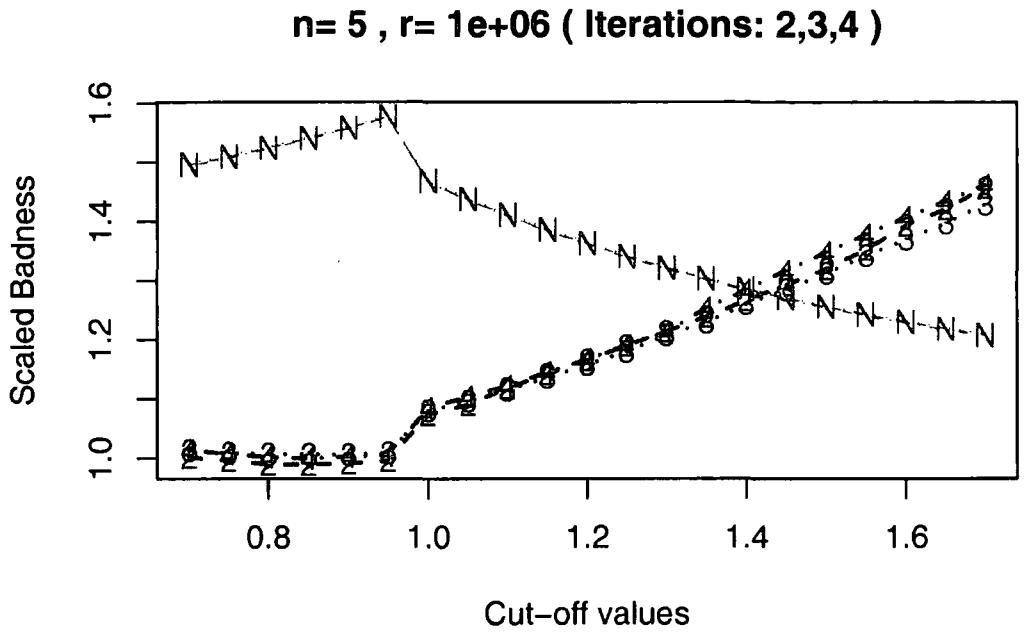


Figure 2.3: Three million samples of size five (Slash vs Normal) - pooled

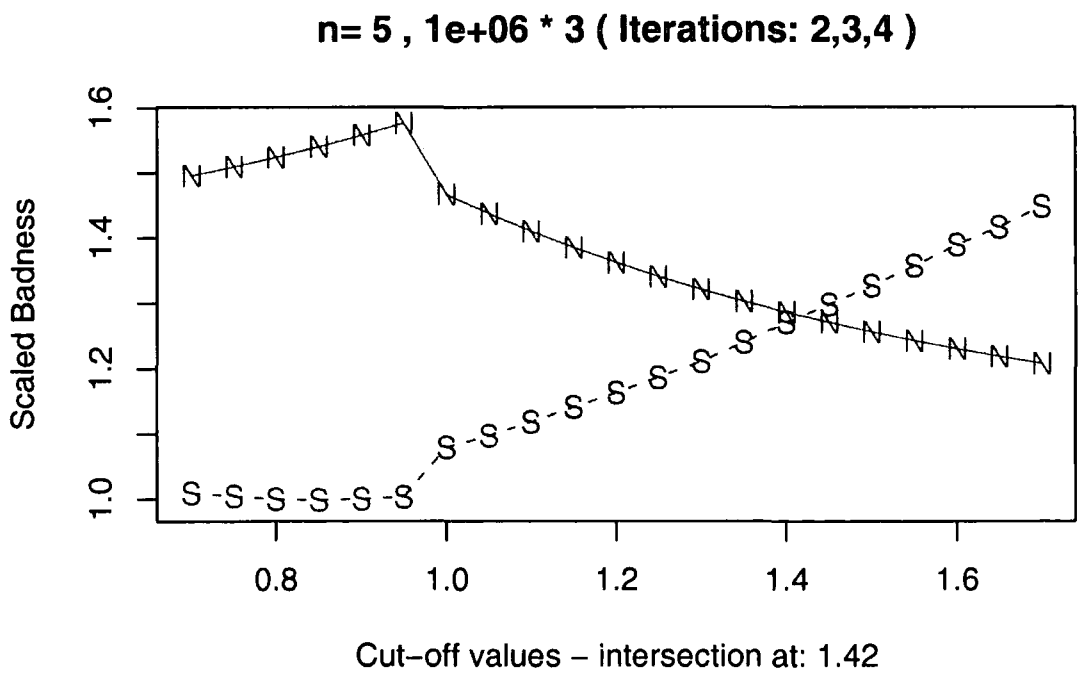


Table 2.6: Initial estimates of optimum cut-off values for Tukey's Method

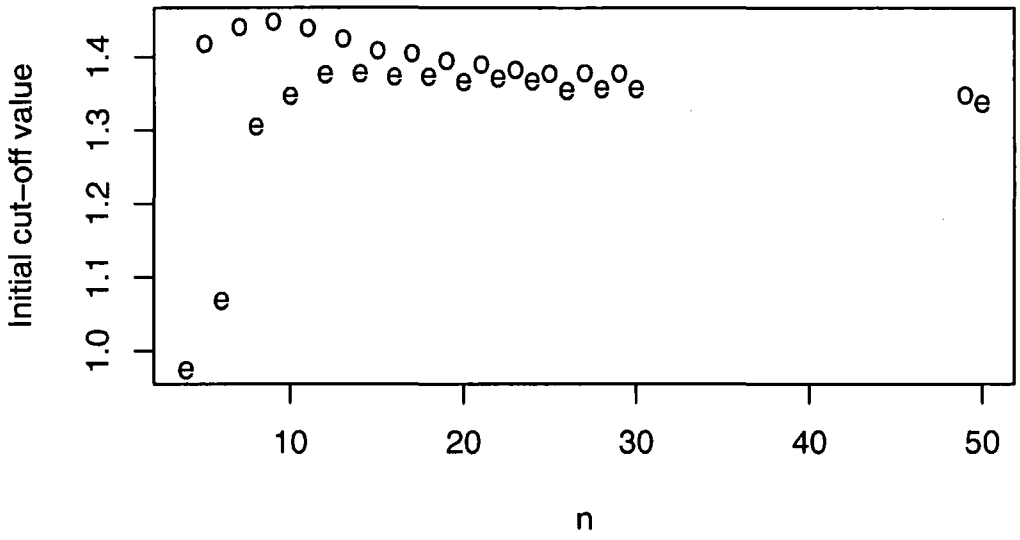
n	$\hat{\gamma^*}'$	N	n	$\hat{\gamma^*}'$	N
4	not-stable	9*500000 + 2*1000000	5	1.4178	3*1000000
6	not-stable	8*500000	7	1.4402	4*500000
8	1.3049	4*500000	9	1.4477	4*500000
10	1.3468	4*500000	11	1.4387	4*50000
12	1.3759	4*50000	13	1.4245	4*50000
14	1.3773	4*50000	15	1.4084	4*50000
16	1.3729	4*50000	17	1.4045	4*50000
18	1.3731	4*50000	19	1.3947	4*50000
20	1.3660	4*50000	21	1.3898	4*50000
22	1.3708	4*50000	23	1.3825	4*50000
24	1.3669	4*50000	25	1.3773	4*50000
26	1.3538	4*50000	27	1.3780	4*50000
28	1.3561	4*50000	29	1.3774	4*50000
30	1.3562	4*50000	49	1.3475	4*50000
50	1.3361	4*50000			

Table 2.6 shows the results of repetition of this procedure for sample sizes up to 50.

Simulations for $n = 4$ and $n = 6$ were not stable enough and no compromise cut-off value could be determined. We split the table into even and odd values of sample sizes because of different behaviours for these two cases. Figure 2.4 shows this difference and it can be seen the difference is decreasing in n.

Stage 3. Finding a more accurate estimate of the optimum cut-off value as well as its confidence interval for univariate data needs another simulation study to approximate the difference between inefficiencies (or scaled badnesses) near the initial optimum cut-off value by a regression line. The details have been discussed in section 2.7.

Figure 2.4: The behaviour of initial optimum cut-off in even and odd sample sizes



The new sets of simulations have been done for a subset of samples sizes which have been studied in stage one. Table 2.7 shows the difference of Slash and Gaussian scaled badnesses i.e. $\hat{\delta}(\gamma_i)$ where $i = 1, 2, \cdots, 21$ and $\underline{\gamma} = \{1.30, 1.31, \cdots, 1.50\}$ for $n = 15$. Each row of the table comes from two separate simulation with 10,000,000 samples (42 simulations in total). The table is followed by the scatterplot and the regression error plots. The plots show a reasonable independence and normality of the errors.

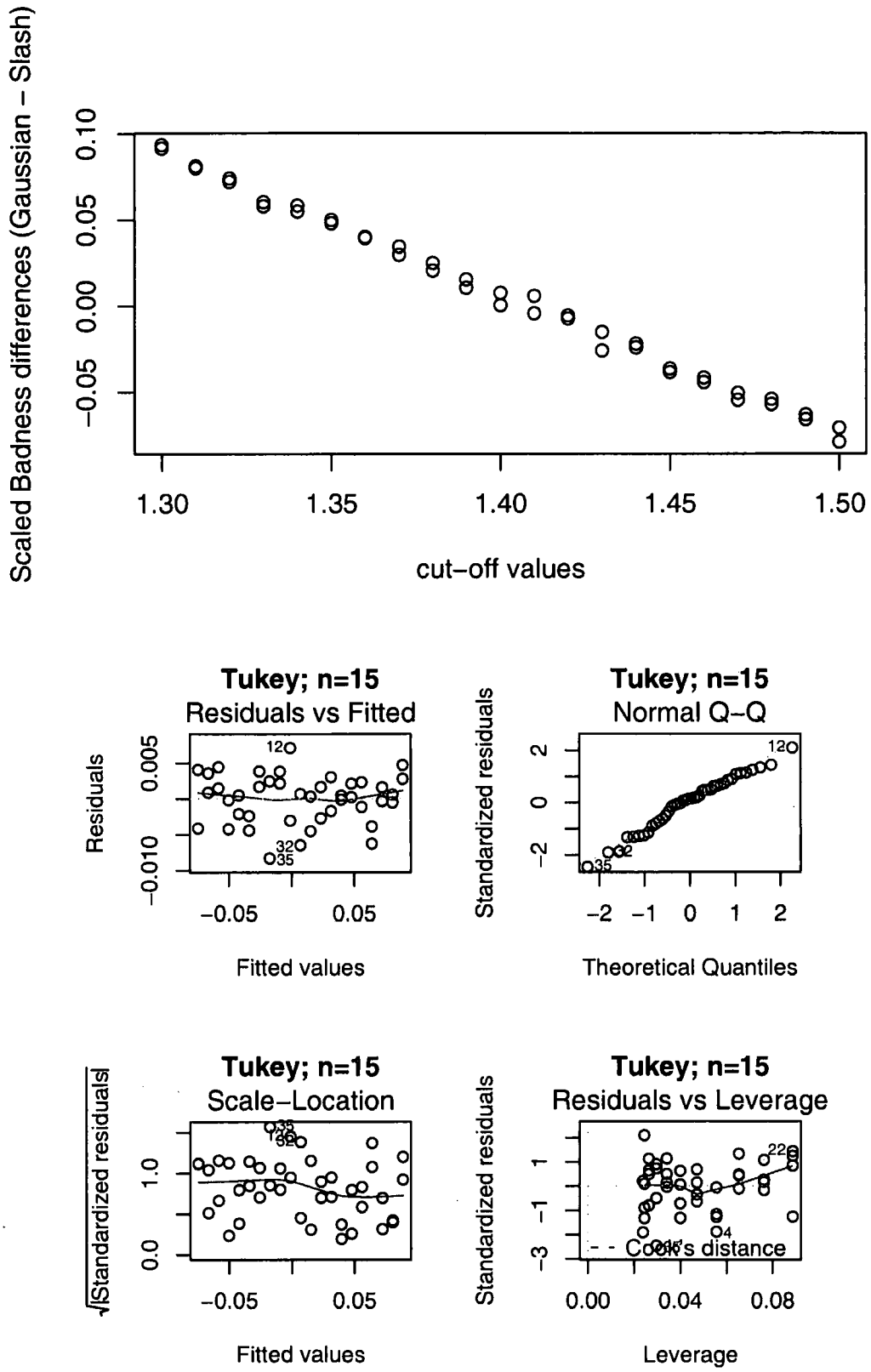
The regression line is $\hat{\delta}(\gamma) = \hat{\alpha} + \hat{\beta}\gamma = 1.1513 - 0.8174\gamma$. Putting $\hat{\delta}(\gamma) = 0$, we have $\hat{\gamma^*} = -\frac{1.1513}{-0.8174} = 1.4085$ which is a more accurate optimum cut-off value to recognise outliers for a sample sized $n = 15$ based on Tukey's method.

To find how accurate the optimum cut-off value is, a confidence interval for $-\frac{\alpha}{\beta}$ in a regression line is used. Despite of the discussion in section 2.7 to consider the uncertainty of minimum badness in Slash samples, table 2.8 contains the confidence intervals without this consideration as the work had been done before the discussion. For instance, using (2.2), the calculated 95% confidence interval for optimum cut-off value when $n = 15$ in the Tukey's method is $(1.4072, 1.4098)$. For some sample

Table 2.7: For $n = 15$, the difference of Slash and Gaussian scaled badness near the initial estimated optimum cut-off

γ_i:cut-off value	$\hat{\delta}(\gamma_i)$:$\hat{B}^G(\gamma_i) - \hat{B}^S(\gamma_i)$
1.3	0.0915 , 0.0934
1.31	0.0811 , 0.08
1.32	0.0739 , 0.072
1.33	0.0579 , 0.0603
1.34	0.0548 , 0.0583
1.35	0.0499 , 0.048
1.36	0.0401 , 0.0395
1.37	0.0345 , 0.0298
1.38	0.0249 , 0.0206
1.39	0.0154 , 0.0106
1.4	0.0076 , 0.0005
1.41	0.0059 , -0.0043
1.42	-0.0073 , -0.0056
1.43	-0.0151 , -0.0259
1.44	-0.0241 , -0.0219
1.45	-0.0384 , -0.0363
1.46	-0.0442 , -0.0416
1.47	-0.0505 , -0.0545
1.48	-0.054 , -0.057
1.49	-0.0658 , -0.0631
1.5	-0.0789 , -0.0707

Figure 2.5: Scatterplot and the residual plots of the regression on data in table 2.7



sizes, the confidence intervals contains the old cut-off values while for some others it does not. Checking for the possibility of needing a quadratic model shows except

Table 2.8: Initial and more exact optimum cut-off values with 95% confidence intervals

n	$\hat{\gamma}^{*\prime}$	$\hat{\gamma}^{*}$	95% CI - L	95% CI - U
5	1.4178	1.3995	1.3802	1.4188
7	1.4402	1.4476	1.4421	1.4530
8	1.3049	1.3007	1.2878	1.3137
9	1.4477	1.4490	1.4456	1.4525
10	1.3468	1.3597	1.3571	1.3622
11	1.4387	1.4300	1.4279	1.4322
12	1.3759	1.3671	1.3644	1.3698
13	1.4245	1.4179	1.4161	1.4197
14	1.3773	1.3651	1.3627	1.3675
15	1.4084	1.4085	1.4072	1.4098
16	1.3729	1.3650	1.3632	1.3669
17	1.4045	1.4000	1.3974	1.4026
18	1.3731	1.3627	1.3606	1.3648
19	1.3947	1.3950	1.3930	1.3970
20	1.3660	1.3659	1.3635	1.3682
30	1.3562	1.3566	1.3545	1.3587

for $n = 7$, all the rest are *not* significant.

2.8 Comparing Modified Z and Tukey's Methods

In this section, we try to compare two methods: *Modified Z-scores* (MZ) and *Tukey*. We understand that the cut-off value suggested by Iglewicz and Hoaglin (1993) is fixed at 3.5 but to do the comparison, a wide range of cut-offs has been applied, something which has not been studied before. Therefore, some simulations have been done to find a compromise cut-off value in each method for a wide range of

sample sizes (5,7-20,30)[3]. The simulations are again based on generation of correlated random numbers from Slash and Gaussian distributions as before. The results of these series of simulation for Tukey's method are naturally different from the simulations explained in previous sections.

To make the two methods comparable, we need to scale them by a single scale which should be chosen by the minimum of two minimum badnesses obtained in two methods. The reason is that if we knew the population type, the best result would be obtained by using the best method with its best cut-off and so inefficiency should be measured relative to that level of badness. The minimum of expected badness in Gaussian population is $1/n$ where n is the sample size, regardless of the method. Therefore, the inefficiencies would only be changed in Slash population.

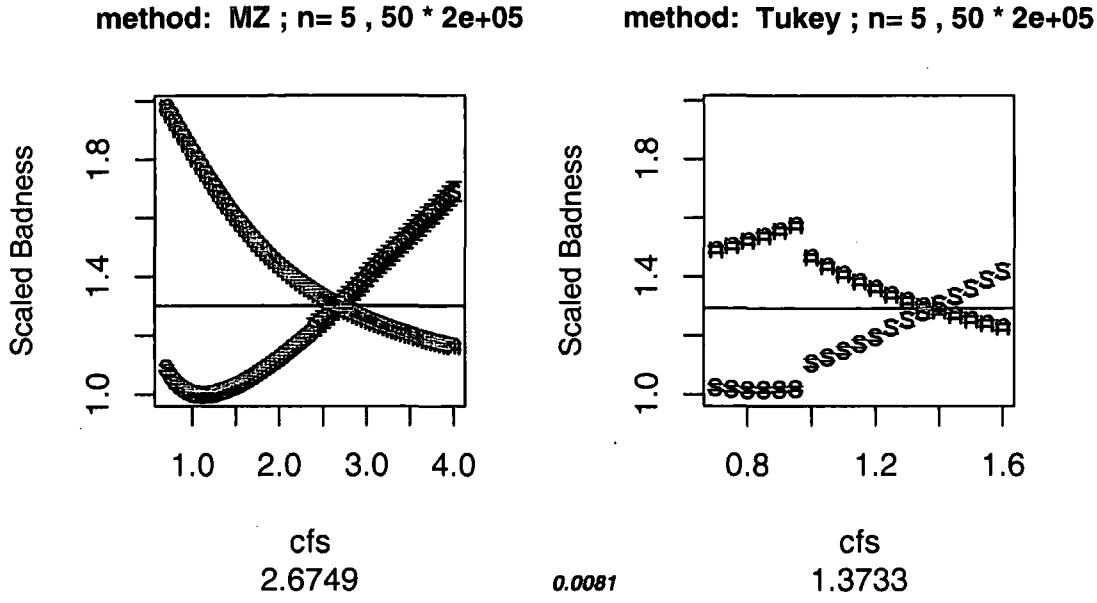
Investigated cut-off values in MZ method are $\{0.7, 0.75, 0.8, ..., 4\}$ which was a fixed cut-off 3.5 in the earlier studies. The set of cut-off studied in Tukey's method is $\{0.7,0.75,0.8,...,1.6\}$. Here is the explanation of the details of calculation for $n = 5$ after running the simulations.

For $n = 5$, the minimum badness by applying MZ occurs at cut-off 1.15 with the amount of 2.1066 while the minimum of badness applying Tukey's method is 2.1402 at the cut-off 0.85. Then raw badnesses obtained by both methods are scaled by 2.1066 which is minimum of the two minimums. The optimum cut-off value for $n = 5$ by this scaling in MZ is 2.6749 and in Tukey's method is 1.3971. The scaled badness or inefficiency using Tukey's method with its optimum cut-off value for $n = 5$ is less than the same calculation using MZ and its optimum cut-off. The difference is 0.0081. See figure 2.6 linked to the above calculations for $n = 5$.

Table 2.9 shows similar calculations for other sample size n. The table splits odd and even n because the behaviour of some columns of the table is separately regular for odd and even values. Although, for both odd and even n, $\hat{\gamma}_S^*$ is almost increasing in terms of n for MZ's method and not for Tukey's method (see figures 2.7 and 2.8). $B_m^{S'}$ decreases as n increases regardless of which method is used or whether n is even or odd (see figures 2.9 and 2.10). Again, in terms of odd or even n's, cut-off values in

[3]We did the simulations for $n = 4$ and $n = 6$ but did not bring the results in the tables and no discussion on them because the simulation results in section 2.7 were not stable for them.

Figure 2.6: The plots of badnesses of Slash and Gaussian samples size $n = 5$ in two methods



each of the two methods are regularly different (see figures 2.11 and 2.12). Finally, the difference between inefficiencies in the two methods, depending on odd or even n, is increasing which means for larger sample sizes, Tukey's method is preferred (see figure 2.13). Except for $n = 8$, Tukey's method is more efficient than MZ.

As another check for $n = 4$, confidence intervals for any individual cut-off values are too big which implies the results of simulation are not stable (see figure 2.14). Furthermore, there is a jump at cut-off one and it makes the function of scaled badness not to be differentiable. It means no confidence interval for the optimum cut-off value could be found at $n = 4$.

Table 2.10 shows the confidence intervals for some n in both MZ and Tukey.

2.9 Miscellaneous

◇ Table 2.11 shows the inefficiency using a fixed cut-off value for all n in each method. Column *diff.* is the difference between badness using the fixed and the estimated optimum cut-off value. We did not take a rule to choose these two fixed cut-off.

Table 2.9: Comparison of two methods and their expected badnesses at their optimum cut-offs

n	MZ			Tukey			expected badness
	$\hat{\gamma}_m^S$	$B_m^{S\prime}$	$\hat{\gamma}^*$	$\hat{\gamma}_m^S$	$B_m^{S\prime}$	$\hat{\gamma}^*$	diff.
5	1.15	2.1066	2.6749	0.85	2.1402	1.3971	0.0081
7	1.2	1.1397	2.6676	0.95	1.1789	1.444	0.0237
9	1.3	0.7939	2.6593	1	0.8224	1.4488	0.0365
11	1.35	0.6115	2.6442	1	0.6263	1.4291	0.045
13	1.4	0.4983	2.6322	1	0.5084	1.4173	0.0526
15	1.4	0.4204	2.6202	1	0.4286	1.4069	0.0582
17	1.45	0.3634	2.6031	1	0.3708	1.4006	0.0634
19	1.45	0.3201	2.5932	1.05	0.3267	1.3926	0.0678
8	1.1	0.9347	2.5592	0.95	1.014	1.3058	-0.0113
10	1.3	0.6853	2.5841	0.95	0.736	1.3583	0.0145
12	1.3	0.5433	2.5803	1	0.576	1.3672	0.0287
14	1.35	0.4513	2.5775	1	0.4724	1.3644	0.0413
16	1.4	0.3865	2.5733	1	0.4016	1.3636	0.0507
18	1.4	0.3378	2.5672	1	0.3502	1.3638	0.0564
20	1.45	0.3	2.5597	1.05	0.3111	1.3654	0.0617
30	1.45	0.1926	2.5353	1.1	0.1984	1.3548	0.0812

Figure 2.7: $\hat{\gamma}_m$ (cut-off corresponding to minimum badness in initial simulation) versus sample size n for modified Z-scores

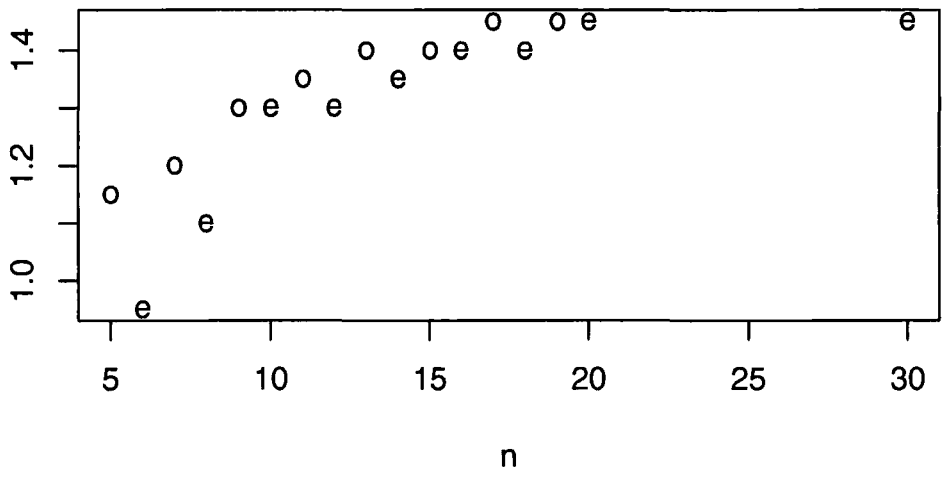


Figure 2.8: $\hat{\gamma}_m$ (cut-off corresponding to minimum badness in initial simulation) versus sample size n for Tukey's method

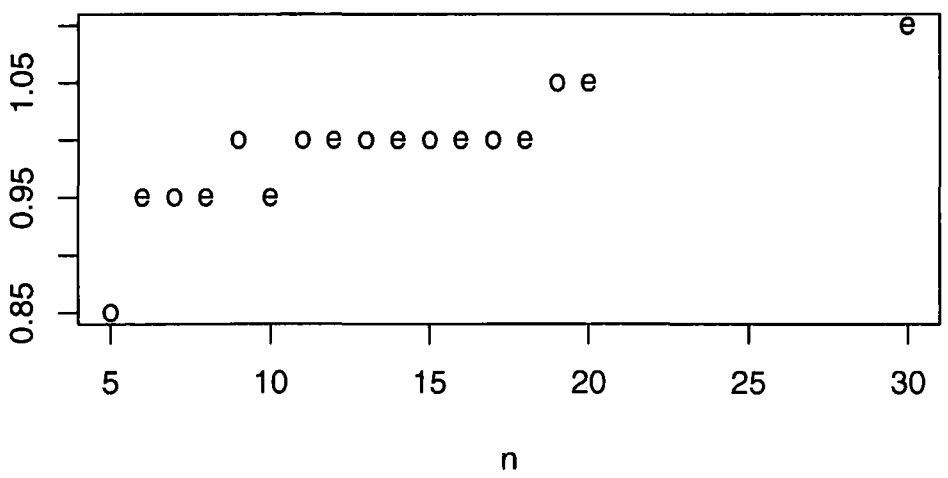


Figure 2.9: $B_m^{S'}$ (minimum badness in initial simulation) versus sample size n for modified Z-scores

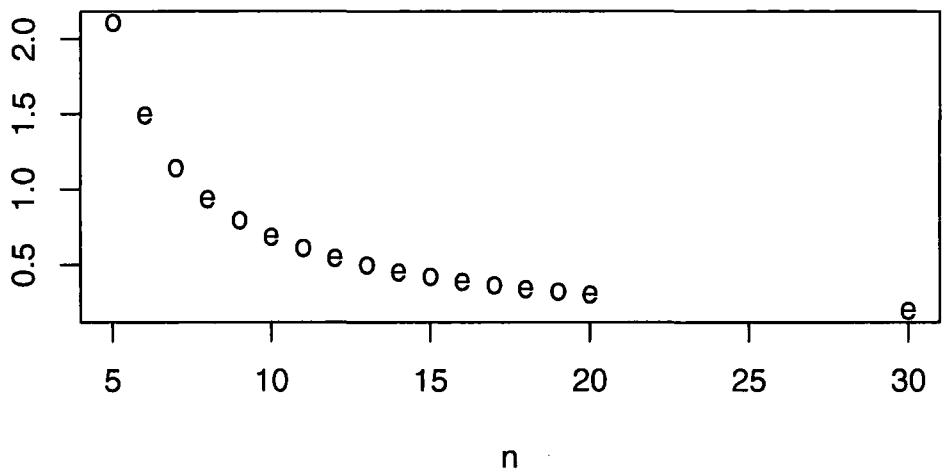


Figure 2.10: $B_m^{S'}$ (minimum badness in initial simulation) versus sample size n for Tukey's method

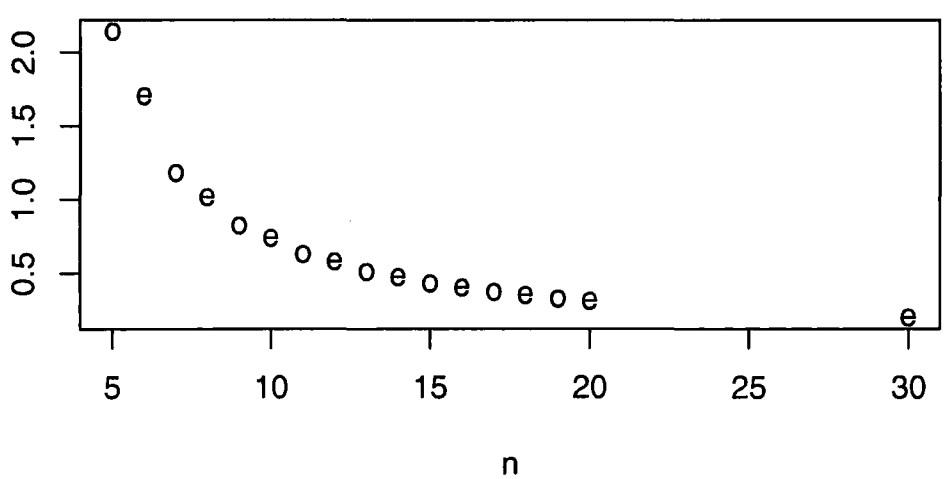


Figure 2.11: $\hat{\gamma}^*$ (optimum cut-of value) versus sample size n for modified Z-scores using the smaller minimum badness to scale

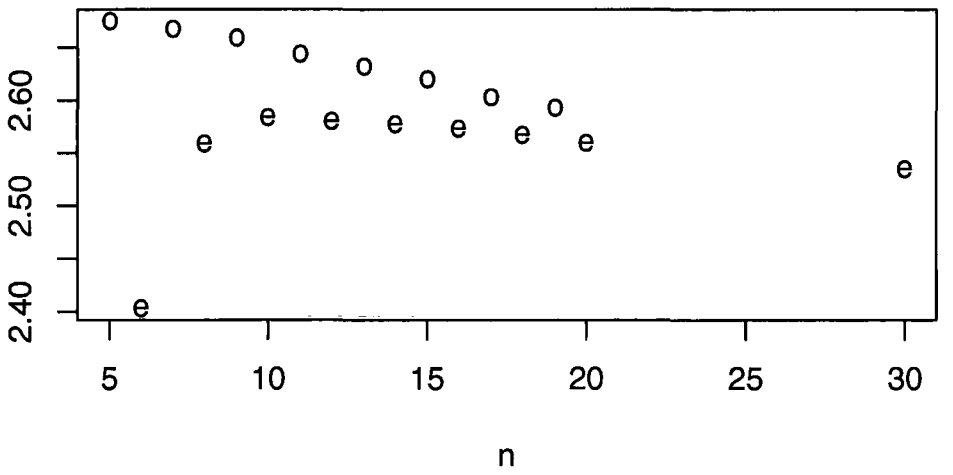


Figure 2.12: $\hat{\gamma}^*$ (optimum cut-of value) versus sample size n for Tukey's method using the smaller minimum badness to scale

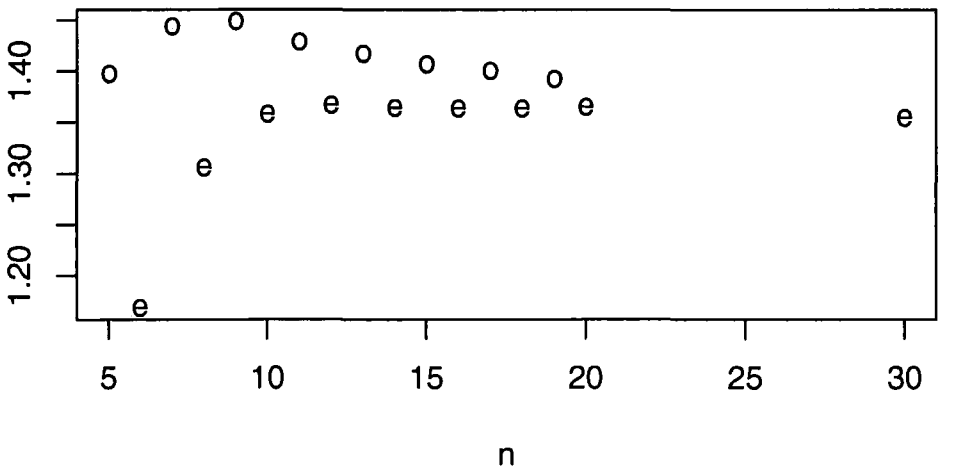


Figure 2.13: The differences between the scaled badness at optimum cut-off value for modified Z-scores and Tukey's method

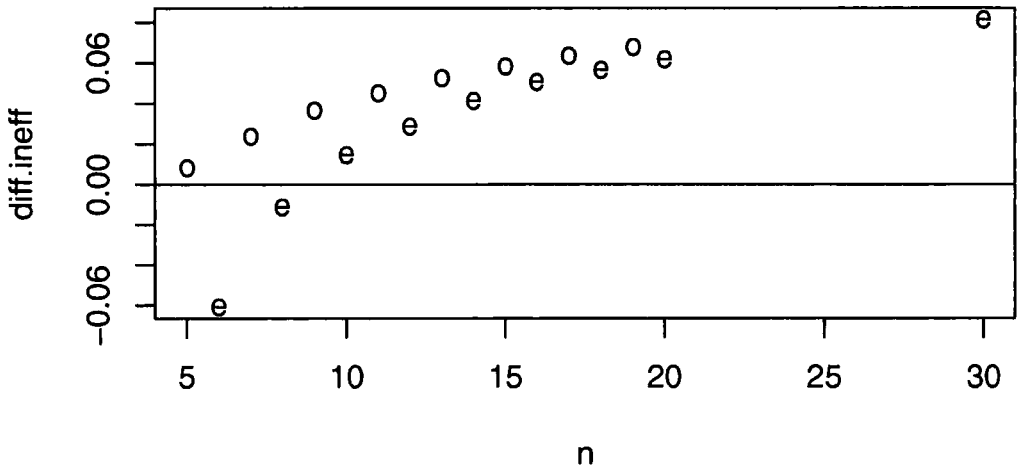


Figure 2.14: For $n = 4$, modified Z-scores and Tukey's method scaled by their own minimum badness with confidence interval at each cut-off

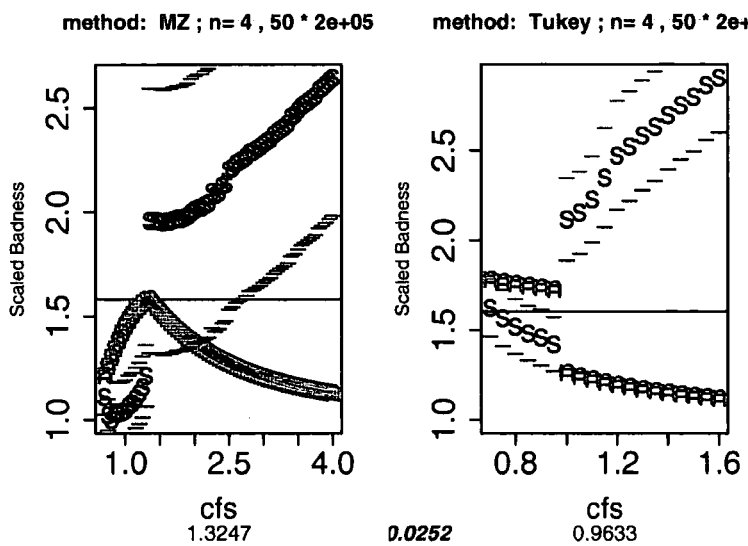


Table 2.10: Initial and the final $\hat{\gamma}^*$ with 95% CI for both MZ and Tukey

n	Modified Z-scores				Tukey's method				
	$\hat{\gamma}^{*'}$	$\hat{\gamma}^*$	L	U	$\hat{\gamma}^{*'}$	$\hat{\gamma}^*$	L	U	
5	2.6761	2.6790	2.6563	2.7018	1.3971	1.3995	1.3802	1.4188	
7	2.6680	2.6786	2.6672	2.6900	1.4440	1.4476	1.4421	1.4530	
8	2.5592	2.5617	2.5529	2.5706	1.3060	1.3007	1.2878	1.3137	
9	2.6592	2.6579	2.6494	2.6663	1.4488	1.4490	1.4456	1.4525	
10	2.5841	2.5802	2.5744	2.5860	1.3586	1.3597	1.3571	1.3622	
11	2.6442	2.6463	2.6414	2.6511	1.4291	1.4300	1.4279	1.4322	
12	2.5803	2.5806	2.5748	2.5865	1.3682	1.3671	1.3644	1.3698	
13	2.6322	2.6290	2.6234	2.6346	1.4178	1.4179	1.4161	1.4197	
14	2.5775	2.5782	2.5734	2.5830	1.3652	1.3651	1.3627	1.3675	
15	2.6205	2.6163	2.6117	2.6210	1.4072	1.4085	1.4072	1.4098	
16	2.5739	2.5747	2.5709	2.5784	1.3642	1.3650	1.3632	1.3669	
17	2.6032	2.6029	2.5986	2.6072	1.4006	1.4000	1.3974	1.4026	
18	2.5674	2.5653	2.5610	2.5696	1.3644	1.3627	1.3606	1.3648	
19	2.5932	2.5926	2.5889	2.5964	1.3926	1.3950	1.3930	1.3970	*
20	2.5598	2.5610	2.5578	2.5643	1.3660	1.3659	1.3635	1.3682	
30	2.5353	2.5346	2.5317	2.5376	1.3551	1.3566	1.3545	1.3587	

Table 2.11: Inefficiency (scaled badness) difference when choosing a fixed cut-off compare to using the optimum cut-off

n	MZ			Tukey		
	Inefficiency		diff.	Inefficiency		diff.
	2.6	$\hat{\gamma}^*$		1.4	$\hat{\gamma}^*$	
5	1.3132	1.3008	-0.0124	1.2866	1.2851	-0.0015
7	1.2689	1.2563	-0.0126	1.2315	1.2160	-0.0155
8	1.2444	1.2338	-0.0107	1.2556	1.2000	-0.0556
9	1.2404	1.2286	-0.0117	1.1925	1.1749	-0.0177
10	1.2168	1.2128	-0.0039	1.1836	1.1611	-0.0226
11	1.2182	1.2090	-0.0091	1.1628	1.1523	-0.0105
12	1.2044	1.1996	-0.0048	1.1569	1.1404	-0.0165
13	1.2026	1.1959	-0.0067	1.1397	1.1334	-0.0063
14	1.1953	1.1898	-0.0055	1.1426	1.1251	-0.0175
15	1.1898	1.1855	-0.0044	1.1197	1.1173	-0.0024
16	1.1882	1.1819	-0.0063	1.1295	1.1118	-0.0176
17	1.1792	1.1785	-0.0007	1.1062	1.1060	-0.0002
18	1.1828	1.1751	-0.0077	1.1180	1.1009	-0.0171
19	1.1730	1.1714	-0.0016	1.0975	1.0942	-0.0032
20	1.1792	1.1697	-0.0095	1.1064	1.0905	-0.0159
30	1.1691	1.1536	-0.0155	1.0768	1.0584	-0.0185

◇ After finding the optimum cut-off values in Tukey's method, a question arises of how many percent of a data set with Slash distribution (having some outliers) will be tagged as outliers using the optimum cut-off values.

To answer this question, a simulation study has been done. For each $n = 5, 7, \cdots, 20, 30$ sample sizes, 10000 random numbers from Slash distribution have been generated. Each generated sample data set will be inspected to detect possible outliers using the optimum cut-off regarding its sample size. Then, the average of 10000 number of detected outliers in each sample size's data sets has been calculated and these averages will be divided by their sample sizes to find the outliers' proportions for each sample size.

Table 2.12 shows the results of three repeated simulation studies producing 10000 Slash random numbers in each repetition and they seem to be consistent. The range of percentages varies between 14% to 18%.

We also did the similar simulation for samples generated from Gaussian distribution to check the optimum cut-off values. It was expected to have very few of outliers. The results in table 2.13 shows the low percentages of outliers especially when the sample size increases:

2.10 The functions in R

- Function `bb0.sim` calling `b.0way.0` has been used for scanning the γ_m. They produce as many samples as required with specific sample sizes from Slash and correlated Gaussian distributions. Then the badness for each element of a set of cut-off values is calculated. Finally, the mean of badnesses in each cut-off value will be saved in separated files. Function `bb0.sim.res` gathers the results of saved files and reports the minimum of badness for each `n`.

- The function `pp0.sim`, can produce a large number of samples from Slash and correlated Gaussian distributions. For a set of cut-off values, badnesses are calculated and scaled by the results of the output of `bb0.sim.res` stored in a variable `bb0simres`. Then using `pp0.sim.res`, a plot of the expected mean of

Table 2.12: Proportion of Slash samples' detected as outliers by Tukey's method

n	Repeat 1	Repeat 2	Repeat 3
5	0.15	0.14	0.15
7	0.15	0.15	0.15
8	0.13	0.14	0.14
9	0.15	0.15	0.15
10	0.14	0.14	0.14
11	0.15	0.16	0.15
12	0.15	0.15	0.15
13	0.16	0.16	0.16
14	0.15	0.15	0.15
15	0.16	0.16	0.16
16	0.16	0.16	0.16
17	0.16	0.16	0.16
18	0.16	0.16	0.16
19	0.17	0.16	0.17
20	0.16	0.16	0.16
30	0.18	0.18	0.17

Table 2.13: Proportion of Gaussian samples' detected as outliers by Tukey's method

n	Repeat 1	Repeat 2	Repeat 3
5	0.08	0.07	0.07
7	0.05	0.05	0.05
8	0.04	0.04	0.04
9	0.04	0.04	0.04
10	0.03	0.03	0.03
11	0.03	0.03	0.03
12	0.03	0.03	0.03
13	0.03	0.03	0.03
14	0.02	0.02	0.02
15	0.02	0.02	0.02
16	0.02	0.02	0.02
17	0.02	0.02	0.02
18	0.02	0.02	0.02
19	0.02	0.02	0.02
20	0.01	0.01	0.01
30	0.01	0.01	0.01

scaled badness will be produced for each group of Slash and Gaussian against the set of cut-off values. The results can be presented as two different plots.

- The function `b.0way` has been used to simulate the univariate data in a short range of cut-off values to be used for finding the confidence intervals. It produces for the desired sample sizes and a given method (modified Z-scores and Tukey's methods).

- `b.0way.res.6.ci` calculates the confidence intervals for the results of simulation near to the initial optimum cut-off value. It uses `b.0way.res.table.1` to retrieve the estimate of minimum badnesses.

- The function `b.0way.summ` does the simulations and calculations needed to produce tables 2.12 and 2.13 (see Appendix B).

Chapter 3

Two Way Factorial Designs

This chapter is about finding the optimum cut-off value to be used for detection of any possible outliers in a two-way factorial design when there is no replication. The method and an old simulation study will be reviewed and the results of an improved simulation study will be presented.

3.1 Introduction

In one of the last papers by Prof. J.W. Tukey (1915-2001), Seheult and Tukey (2001) introduced a method of outlier detection and robust analysis in a factorial experimental design. The basic method for identifying outlying residuals is the same as the third method in univariate data, subject obviously to some changes for analysing a table rather than a vector of data. The cut-off value used in that paper was 1.5 which was based on Fowlkes et al. (1981) and Al-Madfai (1994). The old simulation studies to determine the cut-off value were based on decomposition of 5 by 4 two-way simulated tables[1] using "lo-median" as the sweep function. A sweep function is a location statistic which can be calculated for and subtracted from a set of data. In some sense it can be considered as removing the centre of the data. In this chapter, firstly, that simulation study has been repeated to ensure comparability with the results of previous simulations.

[1] A simple two factorial experimental design without replication

The sweep function in Seheult and Tukey (2001), for an even number of values, is a new concept of median named "fibian". Also, the method in the paper is intended to be used for a generalised factorial experimental design which may have a different number of factors as well as a different number of levels in each factor. The second part of this chapter is another simulation study for a wider range of numbers of levels in a two-way factorial design in order to find a relation between the cut-off value and the number of levels of a two-way factorial design. The new simulation series has been improved in five ways; a) using a similar sweep function to "fibian" named "NE.median" (Johnson 1989; page 13) but easier to use, b) a unique result for the decomposition which is a more user friendly decomposition, c) adding random row and column effects to the generated tables in the simulation, d) censoring Slash random numbers at 80 to control the behaviour of badness and e) using a better approximation for reference values.

The results of the new simulation study are also followed by confidence intervals for cut-off values.

3.2 The method

The method used to distinguish outliers in univariate data may be generalised for a two-way factorial design as follows:

- In chapter 2, calculation of the residuals by subtracting the lo-median from the observations, can be called a "decomposition" which decomposes the data into lo-median and residuals. A simple two-way table can also be decomposed into four components which are the overall parameter, row effects, column effects and the residuals.

- A similar rule must separately be followed for the three non-trivial components effects to distinguish the possible outliers among rows, columns and the residuals.

- To find the best cut-off value to use for each of the three components (rows, columns and interaction), three measures of badness have to be considered.

3.2.1 Decomposition

Decomposition is a method to separate the main effects, interactions (if there are) and residuals in data from a designed experiment. The simplest decomposition is based on the mean as a measure of centre. Using the following example, a mean-based decomposition for a 2^2 factorial design will be explained. The procedure is also called "mean polish".

Example 3.1: Here is the data for a 2^2 factorial design:

factor 1	level 1	level 1	level 2	level 2
factor 2	level 1	level 2	level 1	level 2
response	8	2	6	3

To do a decomposition, we have to make a crosstabulation of the data and add a row and a column of zero to it:

	1	2	
1	8	2	0
2	6	3	0
	0	0	0

Then, starting from rows or columns (as convenient), calculate the mean of main part of crosstable for each row or column, subtract the mean from the elements of the main part and add the mean to the appropriate element of the extra row or column. In this example, we start with the columns (the second factor):

8	2	0		1	-0.5	0		0.75	-0.75	0.25
6	3	0	→	-1	0.5	0	→	-0.75	0.75	-0.25
0	0	0		7	2.5	0		2.25	-2.25	4.75

Repeating the procedure will not make any changes for a mean-based decomposition on a balanced complete factorial design. As well as converging with just one step, a mean-based decomposition does not depend on whether one starts with rows or columns.

In a median-based decomposition, a more resistant sweep function is used but for an even number of values, there is more than one possible value to choose as a

median. In fact, for an even number of values, any number between the two central values can be presented as median.

Fowlkes et al. (1981) and Al-Madfai (1994) have used lo-median when the number of values is even. The lo-median is the lower of the two numbers in the centre for an even number of values. We follow the explanation of rest of the method by polishing an example of 5x4 two-way table using lo-median as the sweep function:

Example 3.2:

-1	-1	11	5
55	-5	-5	0
1	2	-2	11
-1	2	0	-2
1	2	-1	0

As before, the first step is to add a sweep-into column and row to the table:

-1	-1	11	5	0
55	-5	-5	0	0
1	2	-2	11	0
-1	2	0	-2	0
1	2	-1	0	0
0	0	0	0	0

One can choose whether to start from columns or rows. The results are not exactly the same but for our purpose, outlier recognition, both would lead to similar results. In this example, we may start with the columns and find out the lo-median of each column and add it to the element of the sweep-into row corresponding to each column (zeros in the first stage) while it has to be subtracted from each element of the column:

-2	-3	12	5	0
54	-7	-4	0	0
0	0	-1	11	0
-2	0	1	-2	0
0	0	0	0	0
1	2	-1	0	0

In the next step, a sweeping will be done on rows:

0	-1	14	7	-2
58	-3	0	4	-4
0	0	-1	11	0
0	2	3	0	-2
0	0	0	0	0
1	2	-1	0	0

The procedure has to be iterated to achieve a stable table:

0	-1	14	3	0
58	-3	0	0	-2
0	0	-1	7	2
0	2	3	-4	0
0	0	0	-4	2
1	2	-1	4	-2

0	-1	14	3	0
58	-3	0	0	-2
0	0	-1	7	2
0	2	3	-4	0
0	0	0	-4	2
0	1	-2	3	-1

As the lo-median for each column and each row is zero, there is no further change by continuing the sweeping. It usually converges after one or two iterations.

To check that the decomposition has been done in the right way, each cell of the original table should be written as the sum of its corresponding values in the sub-tables and the common value in the right-down corner of the final table. For instance:

$$55 = 58 + (-2) + 0 + (-1)$$

In general, if a_{ij}'s are the original values of the given two-way table, and r_{ij}'s, c_j's and r_i's are the residuals, column and row sub-tables, respectively, provided by the decomposition and let m to be the common value of the decomposition:

$$a_{ij} = r_{ij} + c_j + r_i + m$$

3.2.2 Detecting possible outliers

For a two-way table, the outliers can be separately tagged in the row effects, column effects and the residuals. By considering the row effects as the residuals obtained in a univariate study, Seheult and Tukey (2001) applied exactly the same routine as applied in chapter 2 for univariate data to detect possible outliers. It is the same for the column effects. For the residuals, the procedure is again the same but the number of residuals to be inspected is slightly different. Assume ν is the number of degrees of freedom of the residuals which is $\nu = (r-1)(c-1)$. The ν largest residuals in size should be inspected in the procedure. If there are fewer than ν non-zero residuals, it is recommended to replace ν by the minimum of ν and one more than the number of non-zeros.

In the example, the number of non-zero residuals is 11 which is less than the number of degrees of freedom which is $(5-1)(4-1) = 12$. According to the recommended rule, the minimum of the degree of freedom and "one plus the number of non-zero's" are both 12. Then, we choose the largest 12 residuals in size. Table 3.1 shows the rest of procedure.

Considering a cut-off value of 1.5 leads to tagging the first two residuals in the table as corresponding extreme values. It means that in the original data, we will flag 55 and 11, respectively a_{21} and a_{13}, as extreme values or outliers. If we consider 1.9 as an alternative cut-off value, the result will be the same. But a cut-off value of 1.1 leads to more than two extremes (58, 14 and 7). Based on the rule, if the cut-off value is 0.9, we have still just three extremes because 0.79 has a larger residual than 0.93 which is greater than 0.9.

Obviously, choosing different cut-off values can have different results. Fowlkes et al. (1981) and Al-Madfai (1994) have done some simulation studies to find an optimum cut-off value for a 5x4 two-way table. We first repeated their simulations for a large number of samples and then carried out further work.

Table 3.1: Example 2. Seheult-Tukey's method

(1) Ordered Absolute Selected Residuals	(2) Reference Values	(3) $\frac{(1)}{(2)}$	(4) Scaled by lo-median of (3)
58	1.94	29.90	6.28
14	1.51	9.27	1.95
7	1.25	5.60	1.18
4	1.06	3.77	0.79
4	0.90	4.44	0.93
3	0.76	3.95	0.83
3	0.63	4.76	1.00
3	0.52	5.77	1.21
2	0.41	4.88	1.03
1	0.30	3.33	0.70
1	0.20	5.00	1.05
0	0.10	0.00	0.00

3.3 Simulation study for optimum cut-off value

The idea of how to find out an optimum cut-off value is to generate many samples from a Gaussian distribution while generating other samples from an alternative distribution expected to have some extreme values. The simulation here generates 50000 data sets for a two-way factorial design from a Slash distribution and then 50000 data sets from a Gaussian distribution correlated to the Slash samples. The samples are decomposed to produce the residual part. Then using a set of cut-off values, the corresponding extreme values will be tagged and be temporarily removed from their sample. If the rank of temporary modified sample is big enough (greater than or equal to the degrees of freedom), the next step will be executed which is to calculate an efficient measure to find the difference between estimates and true values. Three measures called by Tukey as the measures of "badness" are the sum of squares of the differences between estimated and true row effects (B_r), the sum

of squares of the differences between estimated and true column effects (B_c) and a linear combination of the last two

$$B_T = rB_c + cB_r$$

where c is the number of levels in the column's factor and r is the number of levels for the row's factor and note that the true (row and column) effects are zero as the described simulation procedure.

The stages to calculate the measures of badness are now shown for our example. The measures will be calculated for a set of cut-off values {0.7,0.9,1.1,3}:

For cut-off value 3, we have just one extreme value and the temporary modified original data (by removing the corresponding extreme) is:

-1	-1	11	5
NA	-5	-5	0
1	2	-2	11
-1	2	0	-2
1	2	-1	0

There are two ways to find out the row and columns effects which are needed to calculate the measures of badnesses: using a standard linear model fitting method and doing a mean decomposition. The two methods yield the same results. When calculating a value to be swept out during the decomposition we ignore completely the removed data. In the current example:

−1	−1	10.4	2.2	0
NA	−5	−5.6	−2.8	0
1	2	−2.6	8.2	0
−1	2	−0.6	−4.8	0
1	2	−1.6	−2.8	0
0	0	0.6	2.8	0

→

−3.65	−3.65	7.75	−0.45	2.65
NA	−0.53	−1.13	1.67	−4.47
−1.15	−0.15	−4.75	6.05	2.15
0.1	3.1	.5	−3.7	−0.1
1.35	2.35	−1.25	−2.45	−0.35
−0.85	−0.85	−0.25	1.95	0.85

and after seven iterations the final decomposed table is:

−2.81	−3.93	7.47	−0.73	2.93
NA	−0.53	−1.13	1.67	−4.47
−0.31	−0.43	−5.03	5.77	2.43
0.94	2.82	0.22	−3.98	−0.82
2.19	2.07	−1.53	−2.73	−0.07
−1.69	−0.57	0.03	2.23	0.57

It is notable that mean decomposition in a complete table does not need any iteration to converge. In other words, at the first step the decomposition is completed. If we remove even one entry from the table, a few iterations are needed to achieve convergence.

Continuing the calculation of badnesses in the example, it is simply calculated by sum of squares of estimated row effects as well as column effects:

$$B_r = (-1.69)^2 + (-0.57)^2 + (0.03)^2 + (2.23)^2 = 8.14$$
$$B_c = (2.93)^2 + (-4.47)^2 + (2.43)^2 + (-0.82)^2 + (-0.07)^2 = 35.11$$

and then

$$B_T = rB_c + cB_r = 5(8.14) + 4(35.11) = 181.16$$

The same routine has to be run for the other cut-off values. For both 0.9 and 1.1 there are three extremes and the modified original table to decompose is:

-1	-1	NA	5
NA	-5	-5	0
1	2	-2	NA
-1	2	0	-2
1	2	-1	0

and the three measure of badnesses can be calculated as above.

For cut-off value of 0.7, there are ten extremes and the modified original table is as the following:

-1	-1	NA	NA
NA	NA	NA	0
1	2	-2	NA
-1	NA	NA	NA
1	2	-1	NA

The linear model's design matrix X corresponding to the modified table has a rank of seven that is less than the eight needed to be a full rank matrix. Then, some effects of the linear model cannot be estimated. In practice, in our simulation study, cut-off values less than 1 usually lead to this situation.

3.4 Results

The simulation study has been done for 50000 samples[2] of 5x4 two-way factorial tables in the first stage to check the results of Fowlkes et al. (1981) and Al-Madfai (1994). Furthermore, for validation, similar simulations have been done for 4x5 tables which should lead to the same results.

We found the optimum cut-off values for each set of effects separately as Fowlkes et al. (1981) and Al-Madfai (1994)[3] did. Table 3.2 includes their results and the presented work.

There are no recorded results for B_r and B_c in Fowlkes's report. It might be because of their emphasis on the total badness (B_T) rather than the other two measures.

For B_T, which is the main measure in the decision problem, the results even using scaling by SqLm in Al-Madfai (1994), are very close.

[2]50000 simulations have been split to five 10000 samples and submitted to "condor", a batch processing system. The result of each split has been stored and then they are gathered to make the result of 50000.

[3]They have also used another scaling named square lo-median scaling or SqLm as a type of within sample scaling. For this scaling, they divide the square root of the ratios of ordered absolute values of each original two-way table to the same number of quartiles of the half positive Gaussian.

Table 3.2: Comparison of old simulations and the present

		B_r	CV(B_r)	B_c	CV(B_c)	B_T	CV(B_T)
Al-Madfai (SqLm)	10000 x 3	1.178 1.93 1.22	0.2929	1.652 1.453 1.855	0.1216	1.69 1.672 1.685	0.0055
Fowlkes	10000 x 5					1.66 1.80 1.68 1.72 1.70	0.0316
This work	10000 x 5	1.38 1.37 1.43 1.37 1.40	0.0183	2.00 1.87 2.24 2.04 2.02	0.0654	1.77 1.68 1.96 1.78 1.73	0.0587

3.5 Weaknesses and Alternatives

i) The first weakness of the above method of polish is that the result is not unique. It depends on the start direction of polishing. There are two different answers if you start the sweeping from the columns or if you start it from the rows. Tukey believed that it does not make effect on distinguishing the outliers. However, having a unique answer would reduce confusion for users of statistical software.

Solution: A unique decomposition for a two-way table[4] can be achieved by doing two polishes, one started with the row and the other one started with the column and then taking an average of the results of row-first polish and the column-first polish. The process is repeated until convergence.

[4]The solution can be used for any higher dimensions of a table

ii) If you have a 2 by c two-way table[5], where $c > 2$, using lo-median to polish we may lose the extreme value in some cases. This is because the lo-median of two numbers is always one of them and sweeping the lo-median makes it zero. It may happen that the value selected by the lo-median is an extreme value but in the residual part of polish it becomes zero and is not considered as an extreme or outlier. This may also happen if lo-median is substituted by hi-median.

Even if mid-median is used, one outlier in a column will spread in the column and both cells might be tagged as outliers.

Solution: An alternative to lo-median/hi-median has been introduced by Seheult and Tukey (2001) named fibian which solves the problem. Fibian is one of lo-median or hi-median depending which one makes the swept-into smaller. Applying fibian to a general design using the method based on the new idea in the next chapter is too difficult. Another alternative named NE-median has been introduced in Johnson (1989; page 13) which is very similar to fibian but it does not use the swept-into value. This alternative is as easy as lo-median/hi-median/mid-median to apply for a generalised new decomposition method. NE-median is the smaller in magnitude of the lo-median and hi-median. If lo-maedian and hi-median are equal in magnitude, NE-median would be set to zero. Using the NE-median, if we first subtract the overall median from the whole table, the results will be very close to the fibian.

In the next section, the simulation study will be based on NE-median as the sweep function after first subtracting the overall median from all cells. If we want to have a decomposition, we need just to add the subtracted median to the estimated total effect which is not necessary in the stage of badness calculation.

iii) In Slash random samples, sometimes a very large number like 700000 may be generated. Such large numbers can have a large influence on the measure

[5]It happens also when there is a r by 2 table, where $r > 2$

of badness so the distribution of badness is very heavily skewed. It leads to an instability in simulation results when the study is repeated. In fact, in the simulation study, we are comparing two type of tables: without extreme values and with some extreme values. In reality, we do not need such large numbers as extremes.

Solution: It is enough for this form of simulation that in the second type of tables there are extremes. The size of extreme are not very important as long as it is considered as extreme comparing to Gaussian. Figure 3.1 shows an overlaid of Gaussian and Slash density functions. It can be seen that any value of Slash greater than 20 in size can be considered as an extreme to Gaussian.
We suggest to truncate any value of Slash random numbers greater than 80 which causes censoring of just %1 of too large and too small numbers, i.e: $P(S < 80) \simeq 0.9950 \Rightarrow P(S < -80) + P(S > 80) \simeq 0.0100$. Truncating Slash leads to change of Slash distribution function to:

$$F_{S_{trM}}(s) = \frac{F_S(s) - F_S(-M)}{F_S(M) - F_S(-M)} \text{ where } M \text{ is the truncated point.}$$

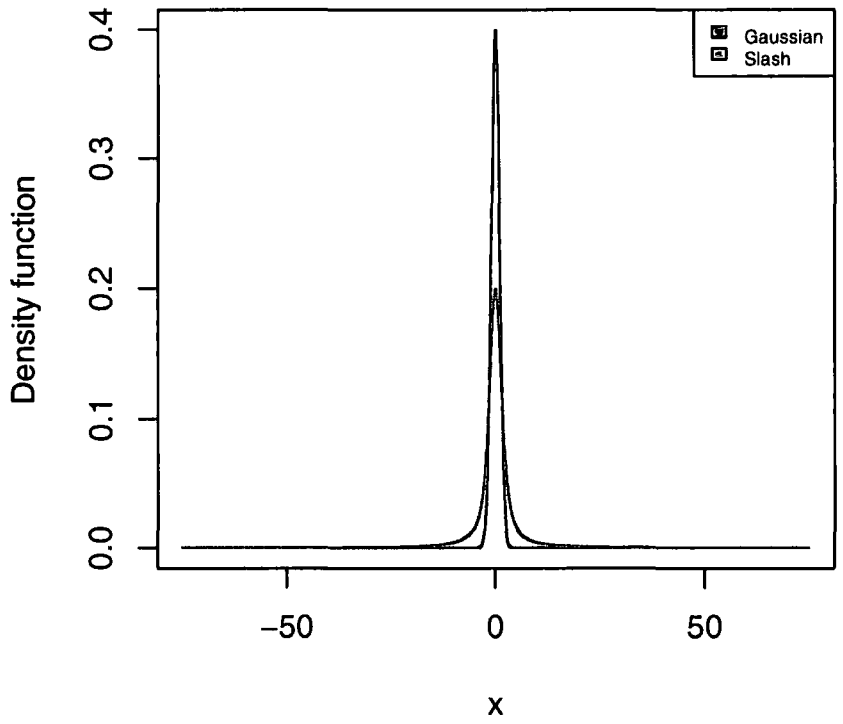
and it should be considered when we generate Gaussian samples by converting Slash samples.

iv) NE.median tries to make the effects as close as possible to zero. Then if the true row effects and column effects are actually zero, it is effectively a bias in the simulation study. In the other words, zeroing row and column effects makes the table almost ready to polish by NE.median.

Solution: In the simulation study, some Gaussian random numbers are added to the samples generated from Slash and converted to Gaussian. One set of Gaussian random numbers is added to rows and another set to columns. It causes the expected row and column effects not to be zero. Then when calculating badness we must subtract those values from the estimated effects before squaring and summing.

v) There are three formulas to approximate the reference values which are m half positive normal expected order statistics q_i where $i = 1, 2, \ldots, m$:

Figure 3.1: Slash density function vs Gaussian



– Seheult and Tukey (2001)

$$2\int_{-\infty}^{q_i} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx - 1 = \frac{m-i+1}{m+\frac{2}{3}}$$

– Al-Madfai (1994)

$$2\int_{-\infty}^{q_i} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx - 1 = 1 - \frac{3i-1}{3m+1} = \frac{m-i+\frac{2}{3}}{m+\frac{1}{3}}$$

Fowlkes et al. (1981) used another form of the second one which is $\frac{i-\frac{1}{3}}{m+\frac{1}{3}}$ where $i = m, (m-1), \ldots, 1$.

Solution: It has an easy solution. A quick simulation study showed the first one is much better.

3.6 New simulation study

A new series of simulations has been done as the final simulations for two-way factorial designs without replication. The basis of the simulation is the same as previous ones but incorporating the improvements in section 3.5.

The first step is to generate rc Slash random numbers restricted to have magnitude less than 80 followed by generating the same number of correlated Gaussian random numbers where r and c indicate the number of rows and the number of columns, respectively, in a simulated two-way table.

Then, for each Slash sample and its correlated Gaussian sample, r ordinary Gaussian will be added to the observations lying in each row of the tables. And similarly, c Gaussian random numbers will be added to each observation on the columns.

A decomposition will be performed for the sample tables using the sweep function "NE.median". The result of decomposition is unique as we polished from both directions (first column and first row) and then average two results and repeat until convergence. It may be called an average based decomposition.

For a given set of cut-off values, the residuals obtained by the decomposition are analysed by Tukey's method to detect possible outliers for a temporary removing from the table.

If the new table without suspected outliers is still a full rank matrix, the measures of badness (row, column and the total) will be calculated for each cut-off and scaled by their minimums. Usually, for smaller cut-off values (about one), the new table without suspected outliers, is not a full rank matrix and then the effects and consequently the measure of badness cannot be computed.

The minimum badnesses for Slash samples have been obtained by the first stage simulations. For Gaussian samples, it can be theoretically calculated by noting that in Gaussian samples it is not expected to tag any data as an outlier. Then keeping all the data leads to the minimum degree of badness. Therefore, the minimum row badness is sum of squares of the usual estimated row effects. It just needed to find its mathematical expectation:

$$
\begin{aligned}
E\left(\sum_{i=1}^{r}(\hat{\alpha}_i)^2\right) &= \sum_{i=1}^{r}\left(E\left(\bar{x}_{i.}-\bar{x}_{..}\right)^2\right) \\
&= \sum_{i=1}^{r}\left(E\left(\bar{x}_{i.}^2-2\bar{x}_{i.}\frac{\sum_{i=1}^{r}\bar{x}_{i.}}{rc}+\bar{x}_{..}^2\right)\right) \\
&= \sum_{i=1}^{r}\left(E\left(\bar{x}_{i.}^2-2\frac{\bar{x}_{i.}^2}{r}+\bar{x}_{..}^2\right)\right) \qquad (3.1) \\
&= \sum_{i=1}^{r}\left(E\left(\frac{1}{c}-2\frac{1}{rc}+\frac{1}{rc}\right)\right) \\
&= \frac{r-1}{c}
\end{aligned}
$$

In similar, the minimum column badness for Gaussian samples is $\frac{c-1}{r}$. Obviously, the minimum for total badness will be $r+c-2$.

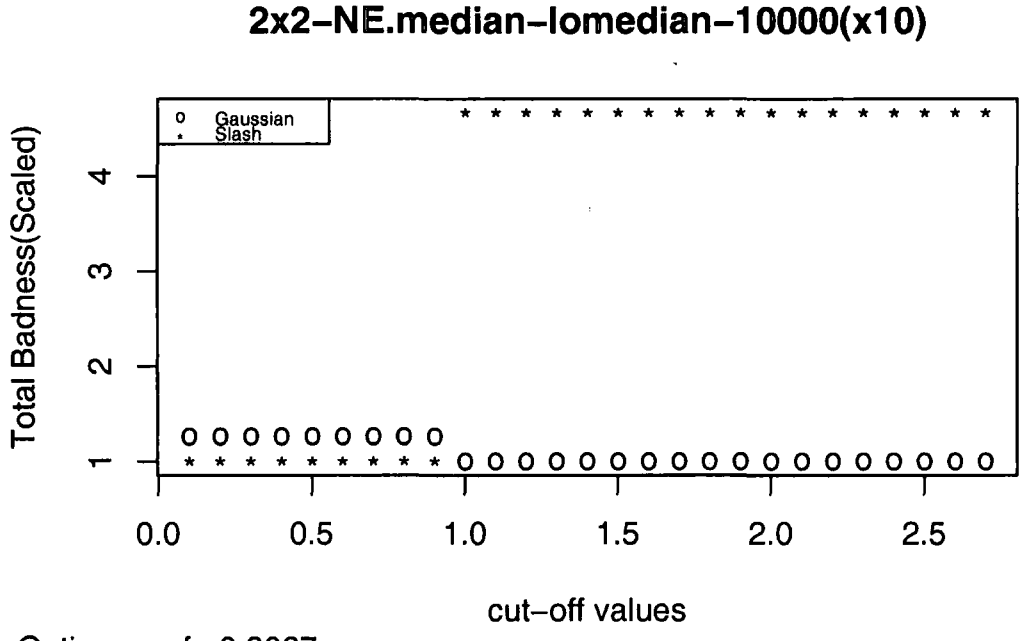
Average and average square of badnesses for each cut-off value will be stored, ready to take account for the final results.

Table 3.3 shows the results of the above described simulation study. The two first columns are the dimensions of the simulated two-way tables. The third column is estimated minimum total badness B_m^S achieved by an independent simulation. The next two columns show the initial and final estimation of the optimum cut-off value. The 95% confidence interval for the final optimum cut-off estimated comes next where the formula 2.1 is used to apply the uncertainty of the estimated minimum badness. Eventually the last column is the p-value of testing the hypotheses of the regression errors to be Normally distributed (see 2.6 for the details of simulation stages and when the regression comes up).

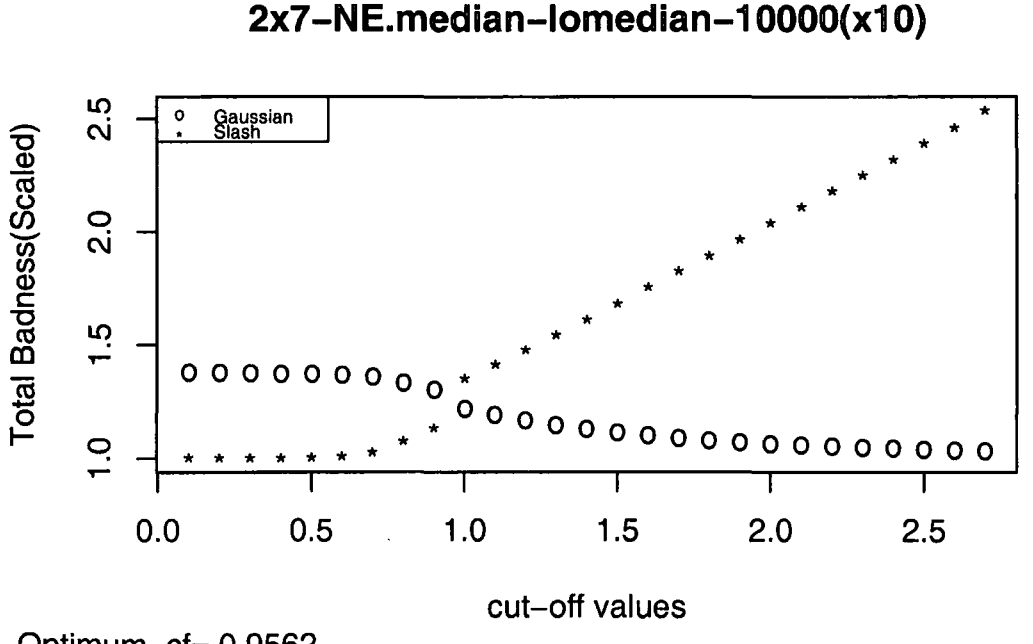
Figures 3.2 and 3.3 show a discontinuity of the measures of badness in 2 by c tables. It means the difference of badnesses is not a continuous function of γ and then it cannot be approximated by a line. As a result, the four last columns of 2 by c rows in table 3.3 have no entries.

If we set aside the two-way tables with at least one two-level factor, the last column of table 3.3 shows that in the regressions applied to find the confidence intervals, the hypothesis of normality for the residuals are not rejected. This is needed in order to trust the regression approximated coefficients. There is an exception on 5 by 6 tables as well as 6 by 5 table that we used to check the algorithms. Then

Figure 3.2: Compromised graphs of a 2x2 and a 2X7 two-way table



Optimum-cf= 0.9067



Optimum-cf= 0.9562

Table 3.3: $\hat{\gamma}^*$ with 95% CI and the p-value to test the Normality of regression errors

r	c	$\hat{\gamma}_m^S$	B_m^S	$\hat{\gamma}^{*\prime}$	$\hat{\gamma}^*$	95% CI-L	95% CI-U	Shapiro.p
2	2	0.1–0.9	26.9494	0.9067	-	-	-	-
2	3	0.1–0.9	35.3009	0.9457	-	-	-	-
2	4	0.3	36.1643	0.9112	-	-	-	-
2	5	0.4	45.2804	0.9417	-	-	-	-
2	6	0.3	53.1658	0.9282	-	-	-	-
2	7	0.3	62.0337	0.9562	-	-	-	-
3	3	1	78.7456	2.0538	2.0537	1.8319	2.2755	0.9989
3	4	1	74.3659	1.6850	1.7830	1.4149	2.1511	0.4128
3	5	1	95.4589	1.8617	1.8736	1.7341	2.0132	0.5092
3	6	1	104.8668	1.7332	1.7190	1.6285	1.8095	0.2404
3	7	1	126.3142	1.8085	1.8264	1.7557	1.8972	0.6950
4	4	1	51.7030	1.3067	1.3034	1.2640	1.3428	0.7706
4	5	1	63.4948	1.4764	1.4864	1.4366	1.5362	0.1472
4	6	1	60.3229	1.3201	1.3190	1.2972	1.3407	0.4783
4	7	1	71.0007	1.4070	1.4080	1.3894	1.4266	0.0619
5	5	1.1	85.2526	1.6938	1.6897	1.6497	1.7298	0.3541
5	6	1	79.9997	1.5197	1.5198	1.4800	1.5595	0.0020
5	7	1.2	96.4786	1.6211	1.6184	1.5748	1.6620	0.5616
6	6	1	68.0780	1.3361	1.3362	1.3205	1.3518	0.4144
6	7	1.1	83.3647	1.4525	1.4537	1.4347	1.4727	0.9870
7	7	1.2	100.1290	1.5484	1.5388	1.5146	1.5630	0.5614

we are not sure if the estimated cut-off for these dimension of tables are accurate. Furthermore, 5 by 6 and/or 6 by 5 tables are the only tables in simulation study that have a unusual behaviour which makes them different to the others. The difference is that with a cut-off value greater than one, removing the suspected outliers makes the rest of design matrix not to be a full rank matrix. We did not consider these few cases in the final calculation. In fact, we reran some more simulations to substitute for them.

The other point to mention is that for all cases the confidence interval includes the initial optimum cut-off value unlike the univariate cases despite our expectation.

3.7 A brief discussion

For any number of levels of a factor, it can be seen that the optimum cut-off value is decreasing for odd number of levels of the other factor and increasing for even number of levels of the other factor. Except if the first factor has three levels the increasing pattern cannot be seen on the even number of levels of the other factor.

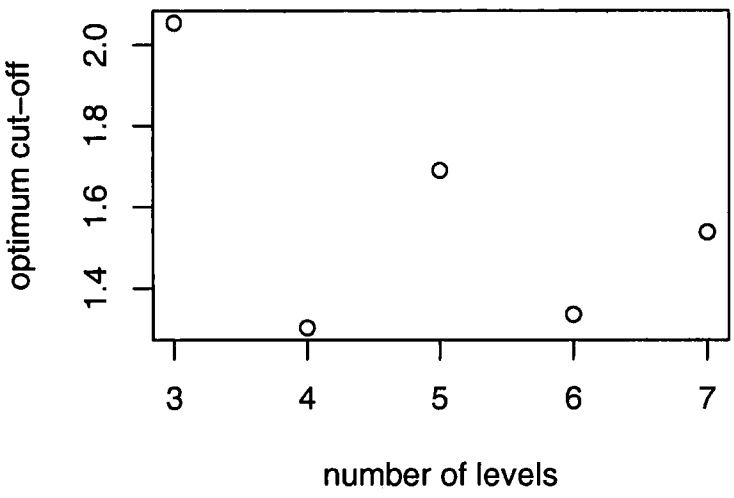
Also we can say that in square two-way tables in odd dimensions the optimum cut-off is decreasing and in even dimensions it is increasing (see figure 3.3).

A more precise pattern might be found if optimum cut-off values for bigger tables will be discovered by simulation.

3.8 Functions in R

- `b.2way.new.2` is to simulate `rr` samples of a factorial experimental design without replication with dimension `lvls`.
- `mat.rank` is the function to find the rank of a matrix (Ripley 2002) which is used after removing the suspected outliers to check if the design matrix is still full rank.
- `b.2way.new.2.res` is to gather the simulation results of 10*10000 of each dimensions of two-way tables. It reports $\hat{\gamma}_m$, initial $\hat{\gamma}^*$, $b_m^{S\prime}$ and $SE[b_m^{S\prime}]$.

Figure 3.3: Estimated optimum cut-off values by simulation in square two-way factorial designs



- `b.2way.new.2.res.m2nd` prepares a batch file in linux to run the second and the third stages of simulation as well running the function of `b.2way.new.2.res` for all simulated dimensions.
- The function `b.2way.new.2.res.mf` is to gather the results of second simulations at $\hat{\gamma}_m$ and reports b_m^S.
- `b.2way.new.2.res.ci` calculates the final $\hat{\gamma}^*$ and its confidence interval in both cases with and without applying the uncertainty of b_m^S (see Appendix C).

Chapter 4

A general decomposition algorithm

In this chapter we introduce a new algorithm to do a decomposition. It is a hierarchical algorithm which is general enough to decompose all types of factorial experimental designs including fractional, incomplete and unbalanced factorial designs. The classical algorithm is not only not directly generalisable but also is not able to cope with absence of one or more interaction effects. The functions in R, provided for the new idea, are able to decompose any type of factorial experimental design using any desired sweep function such as mean, lo-median or NE-median. Recall from chapter 3 that the order of polishing affects the results when the sweep function is not the mean. The functions provided for the new algorithm use the averaging rule which leads to a unique answer.

Some examples have been used to test the algorithm such as factorial designs including unbalanced and fractional factorial designs, incomplete block and Latin square designs. The test is based on three points:

- The results to be the same as the R function `lm` when a mean-based polish is performed
- For any sweep function, the result should be a decomposition
- The result of applying the sweep function on the estimated effects of each term is *zero*

The new decomposition works based on the incidence matrix X_I and two vectors named "effects index" and "levels index". Therefore, producing the incidence matrix X_I for the given model or data frame is the first step which is not completely provided in many statistical software. Most statistical software gives a model matrix which corresponds to a reduced parameterisation for which there are well-defined least squares estimators; often it is a treatment contrasts form of the incidence matrix. The incidence matrix is briefly discussed in Searl (1971; page 166) and Seber (1977; page 72) . In fact, each row of the incidence matrix determines the desired effects to be involved for each response in the model. The function to produce the incidence matrix and desired vectors is also provided in the present work.

4.1 An introduction to the new algorithm

Consider a table of data collected by an experimental design. A model $\underline{y} = X_I\underline{\beta} + \underline{\epsilon}$ may be fitted to the data and the vector of parameters $\underline{\beta}$ estimated. The parameters can be divided into three major parts; the overall mean, main effects and the interactions. Interactions may also be partitioned to the different orders. They all can be easily estimated by applying a least squares solution with desired contrasts. Another way to compute the parameter estimates, when the effects of each term are constrained to sum to zero, is to do a mean decomposition. One method to do a decomposition for a complete factorial design without replication has been introduced in section 3.1. In the example 3.1, the marginals of the result table are the estimated main effects and the right-bottom corner is the estimated total mean. Also, the body of the result table of mean decomposition is the residuals in the model. This method of decomposition can only be used for some types of experimental designs without replication. The new algorithm to decompose hierarchically a table of data in this chapter is applicable for most types of factorial experimental designs. Some motivation for this type of decomposition comes from a paper of Wilkinson (1970) who introduces an alternative approach to computing least squares estimates using sweep operators instead of the usual linear model solution.

4.1.1 A simple example of the new method

Before giving a general explanation of the algorithm, let's perform a simple mean decomposition on the data in example 3.1.

Example 4.1: Using the data in example 3.1, the response vector, the incidence matrix and the vector of model's parameters are as follow:

$$\underline{y} = \begin{pmatrix} 8 \\ 2 \\ 6 \\ 3 \end{pmatrix}, X_I = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix} \text{ and } \underline{\beta} = \begin{pmatrix} \mu \\ \alpha_1 \\ \alpha_2 \\ \beta_1 \\ \beta_2 \end{pmatrix}.$$

In this example, we have just the main effects and the overall mean to be estimated and no interaction. Residuals also will automatically be estimated. In the new approach, decomposition is by estimating the effects using their corresponding columns in the incidence matrix. Before starting the procedure, we put the response values into the residual container as initial values:

$$\hat{\epsilon}_1^{(0)} = y_1 = 8, \hat{\epsilon}_2^{(0)} = 2, \hat{\epsilon}_3^{(0)} = 6, \hat{\epsilon}_4^{(0)} = 3.$$

The aim is to find and take out the effects in the model from these containers.
In general, the procedure starts with the highest order of interaction but no interaction exists in our model. Then we need to start with the main effects. Usually, different ordering of selecting main effects makes difference in the results. However, if the sweep function is mean, as in this example, different orderings lead to the same results. Let us start with α_1 which corresponds to the second column of X_I. A subset of residuals (at the first stage the same as responses) corresponding to the ones in this column is selected (8,2). Their mean is calculated, reserved as the first step of calculating $\hat{\alpha}_1$ and subtracted from the corresponded residual containers. Then it does the same for $\hat{\alpha}_2$. The results are

$$\hat{\alpha}_1^{(1)} = \frac{8+2}{2} = 5, \hat{\epsilon}_1^{(1)} = 8 - 5 = 3, \hat{\epsilon}_2^{(1)} = -3,$$

$$\hat{\alpha}_2^{(1)} = 4.5, \hat{\epsilon}_3^{(1)} = 1.5, \hat{\epsilon}_4^{(1)} = -1.5$$

where the notation (1) means the first stage of decomposition.

The same routine will be done for the other main effects using the updated residuals leaving another updated version of them:

$$\hat{\beta}_1^{(2)} = \frac{3+1.5}{2} = 2.25, \hat{\epsilon}_1^{(2)} = 3 - 2.25 = 0.75, \hat{\epsilon}_3^{(2)} = -0.75,$$

$$\hat{\beta}_2^{(2)} = -2.25, \hat{\epsilon}_2^{(2)} = -0.75, \hat{\epsilon}_4^{(2)} = 0.75.$$

At the end of this stage, estimation of residuals has been completed. The procedure continues to complete the estimation of main effects and the total effect. In the next stage, the column of X_I corresponding to μ, which is the first column, will select the corresponding estimates so far of each main effects. If we start with the first main effect, then

$$\hat{\mu}^{(3)} = \frac{5+4.5}{2} = 4.75, \hat{\alpha}_1^{(3)} = 5 - 4.75 = 0.25, \hat{\alpha}_2^{(3)} = -0.25$$

Continuing with the other main effect does not change anything because of the mean of 0. Therefore, the final estimates are

$$\hat{\mu} = 4.75, \hat{\alpha}_1 = 0.25, \hat{\alpha}_2 = -0.25, \hat{\beta}_1 = 2.25, \hat{\beta}_2 = -2.25$$

$$\hat{\epsilon}_1 = 0.75, \hat{\epsilon}_2 = -0.75, \hat{\epsilon}_3 = -0.75, \hat{\epsilon}_4 = 0.75.$$

4.1.2 An approach to generalise the new method

In the case where there is replication in the two-way factorial design, the two-way interaction effects may also be estimated. In this situation, the X_I matrix has some more columns corresponding to the two-way effects. The procedure to decompose now has to be done in three stages starting from the two-way interaction effects and continuing by the main effects and eventually followed by the total mean as before.

The other issue is to find the residuals. An elegant approach is to augment the X_I matrix by an identity matrix of size of the number of responses. The augmented matrix might be called X_I^*, i.e.

$$X_I^* = (X_I \mid I)$$

Assume that, in a two-way table with replication, we want to estimate the residuals as well as the effects. Performing a decomposition in four stages leads to the right

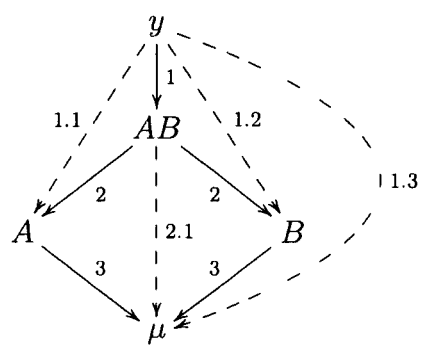
consequences. In this situation, the decomposition starts from the new columns in X_I^* which correspond to the residuals. It continues in the next stage with the columns of two-way interactions and so on. Of course, in practice, we always need to estimate the residuals as they are needed to tag the outliers.
Even if two-way interactions are not requested or they are not applicable, the X_I^* can be constructed similarly and there are just three stages to get the results.

4.2 The algorithm of the new method

Example 4.2: Consider a two-way factorial design with replication. Then the interaction is also eligible to be estimated. Suppose A and B are the two factors and let AB denote the interaction. Figure 4.1 shows the hierarchical levels for the elements of the full model $y_{ijk} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \epsilon_{ijk}$ where $i = 1, \cdots, r$, $j = 1, \cdots, c$ and $k = 1, \cdots, n$ and r,c and n are the number of levels of factor A, number of levels of factor B and the number of replications in each cell, respectively. Column l^* shows the level numbers and column t^* is a numbering for the terms. With the column l^*, one may explain the hierarchical procedure so that beginning from the highest level each term(s) of the higher level should be swept onto the term(s) in all possible lower levels. We will use index of terms (t^*) later on in the explanation of the general algorithm. The procedure may be repeated till convergence.

Figure 4.1: A graphical representation of the hierarchical algorithm to decompose a complete two-way factorial design with model y~A*B

Index of levels (l^*)	Index of terms (t^*)	Terms
3	4	
2	3	
1	1, 2	
0	0	

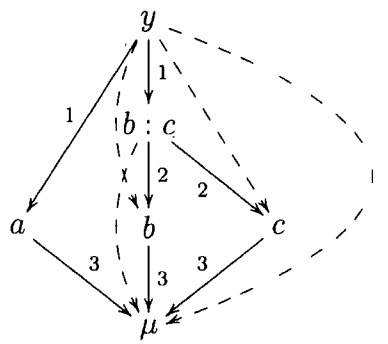


When there is a sweeping onto more than one term in the same level, the ordering should be considered. A strong suggestion to have a unique result is that when sweeping onto several terms at the same level, all possible orders should be applied and then head to the next stage by taking the average of the results of them.
Therefore, the algorithm is a sweeping from top level to the lower ones as the arrows show. First, y will be swept onto AB, main effects and the total effect. Then, the sweeping from AB onto the main effects and the total effect is run. Eventually, each main effect will be swept onto the total effect. Therefore, when AB is swept onto A and B, one might sweep first onto A or first onto B. As it said earlier, the answer is not always the same but the suggestion is to apply both ways and take the average of the two results.
If all the upper levels are complete, continuing to sweep onto the lower levels has no effect but we keep this as part of the algorithm to prevent missing some terms where higher order interactions are omitted from the model. Dashed arrows show the sweeps which have no effect on the results. The next example is a model where an interaction has been omitted.

Example 4.3: The following graph shows the necessary elements of the hierarchical sweeping down for a model of `y~a+b*c`. Again, there are some sweeping shown by dashed arrows which are a part of the algorithm but have no effect in this example. We did not number dashed arrows to prevent losing the clarity of the graph.

Figure 4.2: The graphical show of decomposition for model `y~a+b*c`

Index of levels (l^*)	Index of terms (t^*)	Terms
3	5	
2	4	
1	1, 2, 3	
0	0	



4.2.1 Some necessary definitions used in the algorithm

- We may consider the overall mean μ as the lowest order among the parameters. Then, main effects are the second lowest order, the two-way interactions are the third and so on.
- When a subset of columns of X_I corresponding to an effect can be expressed by a sum of the subset of columns of X_I corresponding to an effect in a higher order, the lower effect is called a child of the higher one.
- In X_I^*, the augmented part is corresponding to y in the model. None of its columns can be expressed by any subset of columns.
- Following the previous definition, y is the highest order term.
- μ is the lowest order term.

4.2.2 Key points in the algorithm

- Applying the given sweep function on the final estimated effects of each term, will give the result of zero.
- The new method is a hierarchical polish by sweeping from the highest order term on its lower order terms (its children!) and then the second highest order term on its lower order terms and it will be continued till reaching to the lowest order term i.e. μ which has no children and cannot be swept.

4.2.3 The algorithm

Some materials need to be prepared for the overall algorithm:

1. Create the incidence matrix X_I as explained earlier.
2. Construct a vector, named "terms' index" and denoted by t, of the same length as the number of columns of X_I with entries of *zero* corresponding to the first column of X_I, *one* corresponding to the columns of X_I allocated for the first main effect, *two* for the second main effect, if exists, and so on.

The numbering will be continued for all two-way and higher interaction effects which are presented in the model. Let s denote the maximum value in t.

3. Construct another vector, named "levels' index" and denoted by l, with the same length as t. The first entry is *zero*, all the entries corresponding to the main effects are *one*, all the entries for two-way interactions (if exists) are *two* and so on. Let g denote the maximum value in l.

4. Construct a third vector of the same length but all entries are *zero*. This is the "coefficient estimates" vector denoted by c.

5. Augment X_I as $X_I^* = (X_I \mid I)$.

6. Augment t to form t^* by adding n extra entries, each equal to $s+1$.

7. Augment l to form l^* by adding n extra entries, each equal to $g+1$.

8. Augment $c^* = (c \mid y)$ where y is the response vector corresponding to the rows of X_I.

The operations of the algorithm are as follows.

Overall algorithm:

Description: A hierarchical sweep starting from the highest level to the second lowest level onto any lower level terms

Action: For each k, indexing terms, descending from $s+1$ to 1, repeat the "complete sweep down" till convergence

Complete sweep down:

Description: Sweeping from a term k onto lower order terms

Action: Let p be the level of term k. For each j descending from $p-1$ to 0, do the "partial sweep down" from term k onto terms of level j

Partial sweep down:

Description: Sweeping from a term k onto all terms of a particular lower level j

Action: For each i appearing in $t^*_{[j]}$ do the "multi-sweep down" from k onto i if i is a child of k. Applying different ordering of i changes the result. Either use a special order or take the average of all possible ordering results

▷ $t^*_{[j]}$ is the subset of t^* corresponding to the entries of l^* which equal j

▷ Term i is a child of term k if variables appearing in term i are a subset of those in term k

Multi-sweep down:

Description: Sweeping from a term k onto a lower level term i

Action: For each m for which $t^*_m = i$ do the "simple sweep down"

Simple sweep down:

Definition: Sweeping from effects in a higher level term k onto the m-th effect (corresponding to a lower level term i)

Action:

1. Find the columns of $X^*_{I[k]}$ which sum to X^*_{Im}. Their corresponding entries in c^* are denoted by $c^*_{[k,m]}$

   ▷ $X^*_{I[k]}$ indicates those columns of X^*_I corresponding to the elements of t^* which equal k

   ▷ X^*_{Im} is the m-th column of X^*_I

2. The requested sweep function is calculated for the elements of $c^*_{[k,m]}$

3. The result is added to c^*_m and is subtracted from each element of $c^*_{[k,m]}$

4.3 Checking the algorithm

Many types of experimental designs can be decomposed by the new algorithm regardless of the number of factors or whether the design is balanced or not or even if it is not a complete design. It also works when there are some missing values in the data. The author has tested the algorithm for the following examples:

- A randomised complete block design (Montgomery 1997; page 177)
- A 5 × 5 Latin square design (Montgomery 1997; page 194)
- A 4 × 4 balanced incomplete block design (Montgomery 1997; page 209)
- A 3 × 3 balanced factorial design with four replications (Montgomery 1997; page 240)

- A $3 \times 2 \times 2$ balanced factorial design with two replications (Montgomery 1997; page 259)
- A 3×3 unbalanced factorial design (Montgomery 1997; page 278)
- A 2^4_{IV} fractional factorial design (Montgomery 1997; page 378)
- The $3 \times 2 \times 2$ balanced factorial design with two replications (Montgomery 1997; page 259) when the model is `~a+b*c`
- The $3 \times 2 \times 2$ balanced factorial design with two replications (Montgomery 1997; page 259) when the model is `~(a+b+c)^2`
- The $3 \times 2 \times 2$ balanced factorial design with two replications (Montgomery 1997; page 259) when there are some missing values among the responses
- A $3 \times 2 \times 4 \times 10 \times 3$ balanced factorial design when the model is `~a+b*(c+d*e)` ("solder" data from "durham" R library, Maths. Dept., University of Durham)

The test is based on achieving the same result of mean decomposition via the new idea and the usual linear model solution.

If the sweep function is different from the mean, they have been tested to be a decomposition and applying the sweep function on the estimated effects of each term to be zero.

4.4 An application of the algorithm

One use of the new algorithm is to produce a list of optimum cut-off values for many types of factorial designs. As an example we have arranged and run a set of simulation studies to estimate the optimum cut-off value for some dimensions of incomplete balanced two-way designs when one observation from each row and each column is missing. All improvement points in 3.5 have been applied in this simulation study, including use of NE-median, averaging for uniqueness and addition of random row and column effects.

Two things have to be mentioned in this simulation study. First, the total measure

of badness is slightly different from the usual (complete) design. In the usual designs, we calculated the total badness by $B_T = rB_c + cB_r$ where B_c and B_r are the column measure of badness and the row measure of badness, respectively. r is the number of rows and c the number columns. In an incomplete design, we may compute B_c and B_r as before by sum of squares of the difference between estimated and true column and row effects after removing the suspected outliers corresponding to a given cut-off.

However, the total badness should be amended. B_c will be multiplied by the number of elements present in each column. Similarly, B_r is multiplied by the number of elements present in each row. One may easily generalise the total badness calculation for an incomplete unbalanced factorial design.

The second notable thing is the minimum expected badness for Gaussian samples. The answer can be obtained as follows:

It has been explained in chapter 2 that the minimum expected badness in Gaussian samples will happen when no observation is removed. Then, the minimum is just the mathematical expectation of each measure of badness. For the row badness,

$$E\left(\sum_{i=1}^{r}(\hat{\alpha}_i - \alpha_i)^2\right) = \sum_{i=1}^{r} Var(\hat{\alpha}_i^2).$$

The variance matrix of the estimated effects can be achieved from the following commands in R:

```
options(contrasts=c('contr.sum','contr.sum'))
summary(lm(mdl))$cov.unscaled
```

The first command is needed to set the constraint of sum to zero to the effects to get the proper answer from the second command. The covariance matrix obtained by the above command has some missing rows and columns; one for each term. They can easily be calculated from the other elements by considering that the missing effect is a linear combination of the other effects in that term. Consequently, because the structure of the incidence matrix makes a symmetry of the order of the effects, the variance of the effects in each term are equal. For the incomplete balanced design, where it is a square design (the same number of levels for both factors i.e. $r = c$),

the variance of any effect in both factors are the same. As a result, the minimum badness for row badness in Gaussian samples are the same as the one for column badness which equals $rVar(\alpha_1)$. Then, the minimum of total badness in Gaussian samples is $2r(r-1)Var(\alpha_1)$ when there is one and only one missing observation in every row and every column.

We could use the similar rule instead of the equation (3.1)

Table (4.1) shows the initial optimum cut-off value ($\hat{\gamma}^{*\prime}$) for some dimensions of an incomplete balanced design. It also contains the estimated cut-off at which the minimum of total badness for Slash samples occurs ($\hat{\gamma}_m$) and also this minimum ($B_m^{S\prime}$).

Table 4.1: $\hat{\gamma}^{*\prime}$ for some dimentions of Incomplete balanced designs - 20000 samples

Dimension	b_m^G	$\hat{\gamma}^{*\prime}$	$\hat{\gamma}_m$	$B_m^{S\prime}$
3x3	5.3333	0.9-1	< 1	84.57
4x4	6.7500	1.5886	1	126.72
5x5	8.5333	1.2733	1	96.13
6x6	10.4167	1.6078	1	150.56
7x7	12.34286	1.3344	1	83.12

It again can be seen that the estimated optimum cut-off values are increasing separately in odd dimensions and in even dimensions. Although, in general, the optimum cut-offs in odd dimensions are less than in even ones.

4.5 The functions in R

- `decomp` decomposes a data set by estimation the parameters defined by a given model using using a given sweep function. The function is in the R library of `robande`.

- `incidence.matrix` makes the desired incidence matrix which is used in the above function. This function is also in the R library of `robande` (see Appendix D).

Chapter 5

Robust analysis of variance

In this chapter the procedure for a robust ANOVA introduced by Seheult and Tukey (2001) will be described and extended to apply to a wider range of factorial designs. Based on the procedure, a library in R has been prepared as one of the outcomes of the present work. Some examples run by the library have been compared with their former analyses.

5.1 The algorithm for the robust ANOVA table

Seheult and Tukey (2001) introduced a procedure to make a robust ANOVA table for a full factorial experimental design. A robust ANOVA table is preferred to the usual one in the presence of one or more outliers or unusual effects in the data. Thus, the first step is to detect the possible outliers which might appear in each model term (obviously except in the total effect). A resistant decomposition is needed to find the resistant effects of any term. Seheult and Tukey (2001) suggested a fibian decomposition. A decomposition includes a sub-table corresponding to each model term. Then, a method is needed to be applied on each sub-table to detect the possible outliers. There are many methods to detect outliers. The method in this procedure and consequently in the provided library is Tukey's method (see sections 2.4 and 3.2 for details). After distinguishing the possible outliers, one may just remove them or they might be substituted by zero as the simplest substitution. Another suggestion is winsorizing the outlier which means that each outlier will

be substituted by the next largest value (in magnitude but the same sign of the outlier) in its sub-table when the smallest outliers is considered as the first largest (in magnitude). In fact, in this procedure, Seheult and Tukey (2001) used a middle way which is half-winsorizing i.e. half of the next largest value is substituted.
After making the substituted table by substituting the suspected outliers, the next stage in the procedure is to do a mean decomposition on the substituted table which will produce what they call inner sub-tables[1]. Robust ANOVA will be based on sums of squares of the estimated effects in the inner sub-tables.
The substituted table differs from the original data and thus the inner tables are not a decomposition of the original data. To make a decomposition table, for each entry corresponding to a suspected outlier in an inner sub-table, they add the difference of the original outlier and its half-winsorized replacement. The resulting tables they call "additive tables" which are a decomposition of the original data.

5.2 R library

The R library `robeda` is provided, based on the procedure introduced by Seheult and Tukey (2001) and generalised for any type of factorial design by the material in chapter four of the present work and by use of sequential ANOVA for non-orthogonal designs. It includes the proper functions to decompose the data collected using a factorial experimental design, to detect the possible outliers and to make the robust ANOVA table. The recipe in Seheult and Tukey (2001) has been also generalised to apply the procedure on full and fractional factorial designs with replication.

5.2.1 Generalisation points and comments

In order to generalise and to program the procedure, some changes are needed. The essential comments and changes are as follows:

[1]Referring to examples 3.1 and 3.2, for a classical decomposition we need to add some zeros to the margins of the table. However, in this case, the extras already exist. Estimating effects using a mean decomposition is made by polishing from that starting point.

- Doing the decomposition using fibian is not easy to program for the general case. A good alternative sweep function is NE-median which has been discussed in section 3.5. The default sweep function in the library is `NE.median` but can be changed by the user to `median`, `lomedian`, `himedian` and/or any other built-in or user-defined function in R.

- It has been discussed in section 3.5 that decomposing the design using different ordering of model terms in each level will lead to different decomposition results. A recommended method to have a unique decomposition is to average the results of different orderings.

- The cut-off value is very important to detect the possible outliers. The default is `1.5` adopted from Seheult and Tukey (2001) but may also be set using tables 3.3 and 4.1 when they are relevant. For now, there is no option in the library to set different cut-off values for different sub-tables in a decomposition.

- In a full factorial design, the estimated effects can be used to directly drive an ANOVA table. However, for an incomplete and/or unbalanced factorial design, a sequential analysis of variance is needed which is not possible or at least not easy to do by using the estimated effects. Therefore, in the library we recompose the data from estimated effects (inner values) and then use `lm` and `anova` to make the ANOVA tables (the standard and the robust).

- There is also an argument named `wins` which can be set to `0`, `1` or `0.5` respectively to apply zeroing, winsorizing or half-winsorizing of detected outliers. The default is `0.5`. Original detected outliers are saved for further studies.

5.2.2 Main functions in the library and the usage

There are 14 functions and 19 data frames in the library `robeda`. The usage of three main functions is explained:

- `incidence.matrix` is a function to produce a complete incidence matrix for any given model formula for a design. The arguments of the function are `mdl` and `dt`. `mdl` is a model formula for the design. The variables in `mdl` must exist

in the current working data image or a data frame containing the variables needs to be defined by dt. Two examples of the usage of this function are (1) incidence.matrix (y ~ r * c) when y, r and c are existed in R working data. r and c must have class "factor"; and (2) incidence.matrix (mdl = 02 , dt = MGDC) when 02 has been already defined by 02 = M ~ (G + D + C) ^ 2 and MGDC is a data frame in R working data and contains variables M, G, D and C. It is required that G, D and C have class "factor".

The output is a list of incidence matrix, indexes of levels and effects and terms' names, respectively named by X, lvls and nterms.

- decomp is the decomposition function which decomposes a data set collected by an experimental design via a given model formula. The arguments are mdl, dt, swp.f and ico. mdl and dt are as described for incidence.matrix. swp.f is the sweep function of the decomposition which may be mean (which is the default), NE.median, lomedian, himedian or some other built-in or user-defined function in R. ico is NULL by default and may be set by the user to be a vector obtained by the function unlist in R which can be applied on a previous decomposition. It is useful when we need to decompose a modified (for example having adjusted for by outliers) former decomposition. The function decomp calls incidence.matrix and then does the decomposition based on the levels' index and effects' index and using the incidence matrix. Three examples of the usage of this function are (1) decomp (y ~ c * r) when y, r and c are existed in R working data and r and c are in a factor class; (2) decomp (02 , MGDC) with the same description for 02 and MGDC in the above; and (3) decomp (02 , MGDC, swp.f = NE.median).

  The output is a list representing the decomposition shown in two forms: (i) as a vector matched to levels and effects indexes; and (ii) as a table (a list of sub-tables) appropriate to display, levels and effects indexes, the incidence matrix and finally terms' names, respectively named by vect, tbl, lvls, X and nterms. The output list has a class of "decomp" which helps to be used in combination with object oriented functions in R. For instance, print.decomp is another function in the library which uses the properties of function print

to display the desired elements of the list.

- `anova.decomp` is the function to do the robust data analysis for a factorial experimental design. The analysis does include a mean based decomposition, a resistant statistic's based decomposition, outlier detection, outlier substitution and robust ANOVA table (see 5.2.1 for details). The arguments are `tabdecomp`, `sclf`, `cf` and `wins`. The first argument is the same list of output from the above function `decomp`. `sclf` is a scale function which is used in Tukey's method when the possible outliers is detected. The default is `"lomedian"` and might be set by the user to `"himedian"` or `"median"`. The default for `cf` is `1.5`. This is the argument which used as cut-off value when the possible outliers are detected. Finally, `wins` is a switch to apply the way of substitution of the suspected outliers. It can be set to any number when the default is `0.5`. According to object oriented properties in R, this function is called by a combination of `anova` function and `decomp` as the following examples: (1) `anova(decomp (y ~ c * r))`; (2) `anova (decomp (O2 , MGDC))`; and (3) `anova( decomp (O2 , MGDC), cf = 1.3)`. In all three examples, arguments have the same properties as those in the other two main functions. The result of this function is a list of the mean decomposition of the data, the decomposition based on the given sweep function, an index table of `0` and `1` indicating the suspected outliers by `1`, the adjusted table by substituting the outliers, inner sub-tables, the additive decomposition table and at last robust ANOVA table, respectively named by `mean.dcmp`, `swp.dcmp`, `outliers`, `half.wins`, `inner`, `add.dcmp` and `Robust.ANOVA`.

5.3 Examples

5.3.1 Dental Gold Data

Table 5.1 shows the data collected as a $8 \times 3 \times 5$ factorial design organised by Xhonga (1971). The response variable is the hardness of the direct gold alloy restoration. Eight types of gold fillings have been used with three methods of condensation by

five dentists.

Table 5.1: Dental gold data - Source: Seheult and Tukey (2001)

Dentist	Condensation Method	Gold Type							
		1	2	3	4	5	6	7	8
	1	792	824	813	792	792	907	792	835
1	2	772	772	782	698	665	1115	835	870
	3	782	803	752	620	835	847	560	585
	1	803	803	715	803	813	858	907	882
2	2	752	772	772	782	743	933	792	824
	3	715	707	835	715	673	698	734	681
	1	715	724	743	627	752	858	762	724
3	2	792	715	813	743	613	824	847	782
	3	762	606	743	681	743	715	824	681
	1	673	946	792	743	762	894	792	649
4	2	657	743	690	882	772	813	870	858
	3	690	245	493	707	289	715	813	312
	1	634	715	707	698	715	772	1048	870
5	2	649	724	803	665	752	824	933	835
	3	724	627	421	483	405	536	405	312

These data have been frequently analysed as in Seheult-Tukey's robust/resistant method (Seheult and Tukey 2001). Table 5.2 shows the fibian decomposition of the data from the paper. NE-median decomposition of the data by the present library `robeda` based on averaging on the same level terms comes in table 5.3.

Comparing to the results in the paper by Seheult and Tukey (2001), the sixth type is tagged (in bold) to be an outlier in the paper while none of the Gold types are identified to be exotic in the averaged NE-median based decomposition. In the paper, only Dentist 5 has been revealed as exotic but Dentist 1 is also tagged in the current calculation. Both calculations have the same result for Condensation method. Furthermore, there are two common distinguished outliers among two-way interactions; one more in the paper and two different ones in the current calculation. Finally, the number of exotic residuals corresponding to individual observations is 19 in the paper while this number is just four in the new analysis which are a subset of the 19. It needs to be remembered that the sweep function in the paper by Seheult and Tukey (2001) is fibian and they choose a specific order of polishing, while the current method is a hierarchical algorithm flexible for any given factorial design model.

However, comparing the polishing when the cut-off value is changed from 1.5 to 1.3 shows many differences (see table 5.4). 11 residuals are tagged as outliers when the

Table 5.2: Fibian decomposition of the data in table 5.1 by Seheult and Tukey (2001)

```
$swp.decomposition
$swp.decomposition$` Overall `
[1] 771

$swp.decomposition$` MGDC$G `
[1]  -9   0   1 -17 -18  95  38  43

$swp.decomposition$` MGDC$D `
[1]  20   1   0 -10 -57

$swp.decomposition$` MGDC$C `
[1]   1   0 -65

$swp.decomposition$` MGDC$G:MGDC$D `
     [,1] [,2] [,3] [,4] [,5]
[1,]    9    0    0  -80  -72
[2,]   32    0  -39   -1    0
[3,]   25   -4   27  -58    0
[4,]  -46   17  -42  149    0
[5,]    0    0   28    0    0
[6,]   26  -60  -30   17    0
[7,]  -38    0    0   33  116
[8,]    0   36  -43 -109    0

$swp.decomposition$` MGDC$G:MGDC$C `
     [,1] [,2] [,3]
[1,]    0    0   56
[2,]    0  -17    0
[3,]   -9   14    0
[4,]    0  -11   25
[5,]   18    0  -16
[6,]   20  -12    0
[7,]    0   38  -11
[8,]    0   51 -172

$swp.decomposition$` MGDC$D:MGDC$C `
     [,1] [,2] [,3]
[1,]    0  -19    0
[2,]   30  -11    0
[3,]  -48    0    9
[4,]    0    0 -146
[5,]    0   27 -208
```

Table 5.2: Fibian decomposition of the data in table 5.1 by Seheult and Tukey (2001) (Cont.)

```
$swp.decomposition$` MGDC$G:MGDC$D:MGDC$C `
, , 1

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    9    0    0    0
[2,]    0    0   39  185    0
[3,]    4  -76    0   96    0
[4,]   63    0  -38 -151    0
[5,]    0   10    0    0    0
[6,]  -26    0   49    0  -58
[7,]    0   66    0  -41  179
[8,]    0    0    0  -47  112

, , 2

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0   30  -15  -11
[2,]  -15   28    0    0    0
[3,]  -30    0    0  -28   47
[4,]    0   32   42    0  -48
[5,]  -89    0 -168   29   29
[6,]  234  149    0  -48    0
[7,]   25  -45    0    0    0
[8,]    4  -67  -40  112    0

, , 3

     [,1] [,2] [,3] [,4] [,5]
[1,]    0  -39    0  173  308
[2,]   45    0  -70 -304  186
[3,]    0  131    0    0  -21
[4,]  -68  -17    0    0   34
[5,]  143    0   34 -227   -2
[6,]    0  -44  -65   53    0
[7,] -155    0   82  203 -179
[8,]  -12   67  138    0    0
```

Table 5.3: Averaged based NE-median decomposition of the data in table 5.1

```
$pres
$Intercept
Intercept
   759.89
$G
    G1     G2     G3     G4     G5     G6     G7     G8
-10.24  -6.75   9.76 -18.53   0.00  75.04  59.50  54.13
$D
    D1     D2     D3     D4     D5
 24.49   4.22   0.00  -5.51 -45.43
$C
    C1     C2     C3
  0.00   0.42 -62.88
$'G:D'
       D1     D2     D3      D4     D5
G1  13.57   0.00   4.21  -67.01 -66.10
G2  22.25  13.35 -38.56   -5.25   0.00
G3   6.96   0.00   8.60  -74.76  -1.40
G4 -52.14  38.29 -18.67   49.03   0.00
G5   0.00 -19.24   3.92    0.00  -7.08
G6  89.92  -4.65 -29.56   10.25   0.00
G7 -48.58 -26.74   0.00   28.31 115.56
G8   0.00  40.26 -57.58 -159.50  24.97
$'G:C'
       C1    C2      C3
G1  -4.12  0.00   50.83
G2  24.12  0.00   -0.83
G3 -15.82  0.00    7.44
G4  -4.37  0.00    1.00
G5   7.63  0.00   -9.00
G6   0.00 18.21  -45.79
G7  -3.29 27.19    0.00
G8   0.00 25.15-196.96
$'D:C'
       C1     C2      C3
D1   0.00 -16.12    6.33
D2  23.50  -2.29    0.00
D3 -19.43   0.00   20.18
D4   0.00   0.21  -16.00
D5   0.00  15.87-151.06
```

Table 5.3: Averaged based NE-median decomposition of the data in table 5.1 (Cont.)

$res.tab

, , C1

	D1	D2	D3	D4	D5
G1	8.42	29.75	-15.31	0	0
G2	0	-15.33	4.72	179.5	-16.83
G3	27.72	-66.56	0	118.44	0
G4	82.67	0	-71.89	-37.5	6.44
G5	0	37	0	0	0
G6	-42.33	0	72.06	54.33	-17.5
G7	0	89.92	-34.67	-46.89	161.78
G8	-3.5	0	-13	0	76.44

, , C2

	D1	D2	D3	D4	D5
G1	0	0	37.72	-20.75	-5.42
G2	-12.17	3.17	0	0	0
G3	-3.39	0	34.33	0	63.89
G4	0	0	19.89	96.5	-47.22
G5	-103.67	0	-151.22	17	28.33
G6	163.17	82.17	0	-45.5	0
G7	28.22	-30.19	0	0	0
G8	22.06	-57.78	0	183.22	0

, , C3

	D1	D2	D3	D4	D5
G1	0	-26.83	0	40.92	**248.97**
G2	60.5	0	-65.06	**-417.67**	134.06
G3	0	116.56	0	-124.94	-95.33
G4	-38.17	-7	0	0	0
G5	116.17	0	30.89	**-377.5**	-79.44
G6	0	-27.83	-1.89	0	6.22
G7	-178.75	0	47.31	49.69	**-270.58**
G8	0	82.33	164.22	-61.17	-70.67

cut-off is 1.3 with eight exotics in two-way sub-tables. The end user should decide on the cut-off value.

Table 5.4: Suspected outliers in Dental Gold Data using two cut-offs, 1.5 and 1.3

```
                    avg -- 1.5
-------------------------------------------
G->
D->D1,D5
C->C3
G:D->G8:D4,G7:D5
G:C->G8:C3
D:C->D5:C3
residuals->res 15,res 29,res 74,res 105

                    avg -- 1.3
-------------------------------------------
G->
D->D1,D5
C->C3
G:D->G6:D1,G3:D4,G8:D4,G7:D5
G:C->G1:C3,G6:C3,G8:C3
D:C->D5:C3
residuals->res 15,res 19,res 29,res 68,res 74,res 81,res 95,res 101,
res 105, res 114,res 118
```

The following results, provided by the default arguments in the library, show the standard and inner mean squares which make a robust ANOVA table. They also includes the percentages of the difference from standard MS to Inner MS. Exotic effects can be also seen in the table for each term.

	DF	Standard.MS	Inner.MS	MS.Changes.Percent
G->	7	31476.85	30519.87	3.04
D->D1,D5	4	54394.10	6213.10	88.58

C->C3	2	298807.60	6611.38	97.79
G:D->G8:D4,G7:D5	28	7457.65	5847.87	21.59
G:C->G8:C3	14	14983.78	5575.51	62.79
D:C->D5:C3	8	32930.12	8357.43	74.62
residuals->res 15,res 29,res 74,res 105	56	9968.89	5905.60	40.76

5.3.2 A randomised complete block design

Table 5.5 contains the gain yield (in Kg) of rice variety IR8 with six different rates of seeding (treatment), collected by a randomised complete block design with four replication (block).

Table 5.5: Grain yield of rice variety IR8 (in Kg) - Source: Bhar and Gupta (2001; page 345)

Treatment	Replication No.			
No.	1	2	3	4
1	5113	5398	5307	4678
2	5346	5952	4719	4264
3	5272	5713	5483	4749
4	5164	4831	4986	4410
5	4804	4848	4432	4748
6	5254	4542	4919	4098

This is the example that Bhar and Gupta (2001) have presented to show how their new developed Cook's statistic works. Two observations have been tagged by their method. The second replication of the second treatment and the fourth replication of the fifth treatment are detected by the developed Cook's statistic.

Table 5.6 shows the estimated effects by an average NE-median polishing. This example could be considered as a 6×4 two-way table. Then from table 3.3 pertaining to a 4×6 two-way table, the cut-off value 1.3190 is used to determine the outliers. As a results, second replication of the second treatment is tagged as an outlier. Furthermore, the fourth replication is revealed to be an extreme replication.

Table 5.6: Averaged based NE-median decomposition of the data in table 5.5

```
$Intercept
Intercept
   4905.5
$c
    c1     c2     c3      c4
 136.0    0.0   13.5 -387.0
$r
    r1     r2     r3     r4     r5     r6
 159.5 -200.0  230.5  -74.5  -57.5    0.0
$res.tab
     r1      r2      r3      r4      r5      r6
c1  -88   504.5       0     197    -180   212.5
c2  333  1246.5     577       0       0  -363.5
c3  228.5     0   333.5   141.5  -429.5       0
c4    0  -54.5       0     -34     287  -420.5
```

The following results also contains the standard and the robust mean squares for the treatment, blocks and the residuals.

```
                  DF Standard.MS Inner.MS MS.Changes.Percent
c->c4              3    648120.3  85208.5          86.852980
r->                5    239666.2 235498.9           1.738794
residulas->res 8  15    110558.4  68667.2          37.890569
```

Bhar and Gupta (2001) remove suspected outliers and find the new F-ratio from the modified ANOVA table. The ANOVA tables after removing the eighth observation which is the second replication of the second treatment, is as follows.

```
> summary(aov(y[-8]~c[-8]+r[-8],ex345bhar))
             Df  Sum Sq Mean Sq F value   Pr(>F)
c[-8]         3 1570221  523407  7.1233 0.003865 **
r[-8]         5 1174433  234887  3.1967 0.039297 *
Residuals    14 1028701   73479
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

Also the following ANOVA table is after removing the eighth and the 23rd observations (the fourth replication of the fifth treatment) which is tagged as outlier by Bhar and Gupta (2001).

```
> summary(aov(y[-c(8,23)]~c[-c(8,23)]+r[-c(8,23)],ex345bhar))
              Df  Sum Sq Mean Sq F value    Pr(>F)
c[-c(8, 23)]   3 1619718  539906 10.3398 0.0009449 ***
r[-c(8, 23)]   5 1445169  289034  5.5353 0.0060088 **
Residuals     13  678809   52216
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

By looking at the residuals mean squares, we understand in the robust ANOVA table MS of residuals is smaller than the standard ANOVA after removing the eighth

observation. However, removing the two observations, this value in robust table is greater. We may wish to compare mean squares column effects in the robust table with the standard ones which is much smaller.

5.3.3 Balanced incomplete block designs

Applying an averaged NE-median polish on a data set adopted from (Montgomery 1997; page 209), it can be seen that the number of non-zero residuals is six which is one more than the degrees of freedom for the residuals in this type of designs (see tables 5.7 and 5.8).

Table 5.7: Original data - Source: Montgomery (1997; page 209)

	Block			
Treatment				
(Catalyst)	1	2	3	4
1	73	74	-	71
2	-	75	67	72
3	73	75	68	-
4	75	-	72	75

Table 5.8: Residuals of the data in table 5.7 obtained by NE-median polish

	Block			
Treatment				
	1	2	3	4
1	1	0	-	-0.5
2	-	0.5	-0.5	0
3	0	0	0	-
4	-0.5	-	1.5	0

Then, following the rule in Seheult and Tukey (2001), the smallest non-zero residuals (in magnitude) would be subtracted from the other five values. Now the

procedure of the detecting outliers will be applied on the five obtained values with three zeros among them. Discrete values in the original collected data for the design may cause this situation in some other real examples. The problem is with three zeroes, the lo-median of the five obtained values is zero and then scaling by the lo-median is not possible. As a consequence, the procedure of detecting outliers cannot be completed.

With a simulation study of 1000 replicates of adding random numbers generated from the uniform distribution, U(0,1), to the above real example, it was revealed that for about 95% of the replications, the number of non-zero residuals obtained from an averaged NE-median polish is six while only in 0.4% of the replications the number of non-zeros are five which is the same as the degrees of freedom (see table 5.9). Thus, for more than 99% of replicates, it was necessary to subtract a non-zero absolute residual from the five largest and this may lead to above problem whenever the data are discrete. Continuing the outlier detection in the hypothetical cases of the simulation, it can be seen in table 5.9 that 16.8% of samples end with detecting the ninth datum to be an outlier. In addition, the first datum is detected as an outlier in 6.6% of samples while more study shows that in 6.4% of samples both the first and the ninth data are detected to be outliers. The conclusion is that if the data are integer, outlier detection cannot be finished while there might be some suspected outliers. In other words, for integer values, Tukey's method may need to be modified, as otherwise there will be an error of calculation during the procedure and it cannot be completed.

An alternative suggestion to keep away from the problem in a real case with the integer data is to use the top 5 actual non zero values without doing the subtraction. Or it might also be suggested to use all of non zero values.

Table 5.9: Number of non-zero residuals, the percentages and detected outliers by a simulation study of 1000 on a BIB design in example 5.3.3

Number of non-zero residuals	5	6	7	8
Percentages appeared	0.4	94.9	4.5	0.2
res 1	0	66	6	1
res 2	0	0	0	0
res 3	0	2	0	0
res 4	0	0	0	0
res 5	0	0	1	0
res 6	0	0	0	0
res 7	1	2	1	0
res 8	0	0	0	0
res 9	4	168	18	2
res 10	1	10	1	0
res 11	0	0	0	0
res 12	0	0	0	0

Chapter 6

Conclusions

This chapter includes a summary of what has been achieved during this work, how the research objectives have been met, what further questions have been raised and suggestions for further work.

6.1 What has been achieved and what is left to do?

In chapter 2, two methods of outlier detection in univariate data have been compared by simulation study and Tukey's method was found to be generally more efficient. A list of optimum cut-off values for a wide range of sample sizes has been provided for Tukey's method. The result shows there is a different pattern for optimum cut-offs on odd and even number of sample sizes but a convergence can be seen for large sample sizes. The list also includes the confidence interval for each optimum cut-off. This list could be improved by repeating the simulation study incorporating the suggestion for improvement in section 3.5 and also using the positively and less biased estimator of minimum badness.

In chapter 3, for a 4×5 two-way factorial design without replication, the optimum cut-off obtained by simulation study has been compared with two old simulations. In section 3.5, some improvements for the simulation study have been suggested and then for a wide range of two-way dimensions, optimum cut-off values with their confidence intervals have been obtained using the improved simulations. Near me-

dian, as a choice of one of the middle values in an even number of data, is used as the sweep function of decomposition. A pattern might be found for the optimum cut-offs but simulations for bigger two-way tables are required.
The measure of inefficiency is based on removing the suspected outliers from the data. Possible alternatives are winsorising, half-winsorising or even fractional winsorising the suspected outliers. They might be more efficient.

In Chapter 4, a new hierarchical decomposition algorithm has been developed which can be used for many types of factorial designs. Decomposing a table helps to do an exploratory data analysis and is the basis for the robust analysis of variance introduced in Seheult and Tukey (2001).
Using the new algorithm, many types of factorial designs with or without replication can be decomposed including incomplete, unbalanced, Latin square and fractional factorial designs. One of the most important achievements of this algorithm is the possibility of choosing the desired interaction effects by defining a model. Using the algorithm, one may find lists of cut-off values for a variety of factorial designs; for example, see section 4.4 for such a list. The function in R for the new algorithm is an average based decomposition with a unique answer. It still needs some work to do the decomposition faster.

In Chapter 5, a robust data analysis based on Seheult and Tukey (2001) has been programmed in R. The collection of functions have been put in a library in R named `robeda` and the intention is to submit to CRAN. With the library, many factorial designs and models can be robustly analysed. The final output of the main function of the library is a joint table of standard and robust ANOVA tables. The elements of the decomposition can be displayed or passed to other functions in R. Object oriented properties of R have been used and the usage of main functions is similar to the other functions.

Paul's rule of two is used in Seheult and Tukey (2001) and should be added to the library. A graphical display of the effects and residuals produced by the decomposition helps interpretation and should also be added.

6.2 Further works

- Redoing the simulation study in section 2.7, to find out a more precise list of optimum cut-offs in univariate data by considering the improvement points in section 3.5 and using B_m^S, introduced in section 2.6, to scale the measure of inefficiency.

- A new approach is to try fractional winsorising of suspected outliers during the simulation study and see if it is more efficient.

- More simulation studies for other sizes of layout may lead to finding a pattern for optimum cut-offs in two-way designs.

- There are still many factorial designs for which optimum cut-off values have not been determined. A study (theoretical or by simulation) is needed to find the best cut-off values for each particular design. It might be added as a table to the library to use automatically for different designs.

- The hierarchical algorithm programmed in R might be improved to do a faster decomposition.

- In the paper of Seheult and Tukey (2001), Paul's rule of two is suggested for application to the robust ANOVA table based on Hoaglin, Mosteller, and Tukey (1991; chapter 11). It might be added in future to the library.

- Trimming is another way of dealing with outliers and it seems a good idea for it to be available in the library.

- Methods other than Tukey's method to detect outliers might also be added in the library. Then the user will be able to select from them.

- When the collected data are integer, adjusting the number of non-zero residuals in the outlier detection method might be modified to avoid the unexpected termination of the procedure. Suggested modifications are 1) not to subtract the smallest non-zero residual when the number of non-zeros are more than the degrees of freedom and/or 2) using all the non-zeros rather than just the

number of degrees of freedom of non-zeros. Changing the residuals a little bit is another suggestion which makes undesired zeros disappear.

Bibliography

H. S. Al-Madfai. Detecting outliers in factorial experiments. Master's thesis, University of Durham, 1994.

V. Barnett and T. Lewis. *Outliers in Statistical Data.* Wiley, New York, third edition, 1994.

J. Besag. On resistant techniques and statistical analysis. *Biometrika*, 68:463–469, 1981.

F. W. Bessel. *Fundamenta Astronomiae.* Nicolovius, Konigsberg, 1818.

L. Bhar and V. K. Gupta. A useful statistic for studying outliers in experimental designs. *Sankhya: The Indian Journal of Statistics*, 63:338–350, 2001.

G. E. P. Box and N. R. Draper. Robust designs. *Biometrika*, 62(2):347–352, 1975.

P. J. Burns. *Aspects of Robust Analysis in Designed Experiments.* PhD thesis, University of Washington, 1988.

P. S. Craig. A private communication, 2006.

C. Daniel. Locating outliers in factorial experiments. *Technometrics*, 2:149–156, 1960.

P. Filzmoser. Robust statistics and R, August 2006. http://www.statistik.tuwien.ac.at/rsr/.

E. Fowlkes, J. McRae, A. H. Seheult, and J. W. Tukey. Unpublished technical report. Received from Dr. A. H. Seheult, 1981.

F. E. Grubbs. Sample criteria for testing outlying observations. *Technometrics*, 2: 27–58, 1950.

F. E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11:1–21, 1969.

F. Hampel. The breakdown points of the mean combined with some rejection rules. *Technometrics*, 27:95–107, 1985.

F. Hampel. Robust statistics: A brief introduction and overview. Technical report, Invited talk in the Symposium Robust Statistics and Fuzzy Techniques in Geodesy and GIS held in ETH Zurich, March 12-16, 2001.

D. M. Hawkins. *Identification of Outliers.* Chapman and Hall, London, 1980.

D. C. Hoaglin, F. Mosteller, and J. W. Tukey. *Fundamental of Exploratory Analysis of Variances.* Wiley, New York, 1991.

P. J. Huber. John W. Tukey's contributions to robust statistics. *The Annals of Statistics*, 30(6):1640–1648, 2002.

P. J. Huber. The 1972 wald lecture robust statistics: A review. *The Annals of Mathematical Statistics*, 43(4):1041–1067, 1972.

P. J. Huber. *Robust Statistics.* Wiley, New York, 1981.

B. Iglewicz and D. C. Hoaglin. *How to Detect and Handle Outliers.* Quality Press, Wisconsin, 1993.

H. Jeffreys. *Theory of Probability.* Clarendon Press, Oxford, 1939.

E. G. Johnson. *Robust Analysis of Factorial Designs via Elemental Subsets and Outlier Sterilization.* PhD thesis, Princeton University, Educational testing service, 1989.

S. Kotz, N. L. Johnson, and C. B. Read, editors. *Encyclopedia of Statistical Sciences*, volume 8. John Wiley, New York, 1988.

D. C. Montgomery. *Design and Analysis of Experiments.* Wiley, New York, fourth edition, 1997.

S. Newcomb. A generalized theory of the combination of observations so as to obtain the best result. *American journal of Mathematics*, 8:343–366, 1886.

D. J. Olive. Applied robust statistics, 2006. http://www.math.siu.edu/olive/cont.pdf.

B. D. Ripley. Re: [R] Rank of a matrix?, 2002. http://tolstoy.newcastle.edu.au/R/help/02a/4608.html.

B. D. Ripley. Robust statistics - M.Sc. program in applied statistics - University of Oxford, 2004. http://www.stats.ox.ac.uk/pub/StatMeth/Robust.pdf.

B. Rosner. On the detection of many outliers. *Technometrics*, 17:221–227, 1975.

B. Rosner. Percentage points for a generalized esd many-outlier procedure. *Technometrics*, 25:165–172, 1983.

P. J. Rousseenuw and A. M. Leroy. *Robust Regression and Outlier Detection.* Wiley, New York, 1987.

S. R. Searl. *Linear Models.* Wiley, New York, 1971.

G. A. F. Seber. *Linear Regression Analysis.* Wiley, New York, 1977.

A. H. Seheult. A private communication, 2006.

A. H. Seheult and J. W. Tukey. Toward robust analysis of variances. In *Data Analysis from Statistical Foundations. A Festschrift in Honour of the 75th Birthday of D.A.S.Fraser.*, Ottawa, 2001. Nova Publishers.

R. E. Shiffler. Maximum z scores and outliers. *The American Statistician*, 42:79–80, 1988.

W. Stefensky. Rejecting outliers by maximum normed residual. *The Annals of Mathematical Statistics*, 42:35–45, 1971.

W. Stefensky. Rejecting outliers in factorial designs. *Technometrics*, 14:469–479, 1972.

A. J. Stromberg. Why write statistical software? the case of robust statistical methods. *Journal of Statistical Software*, 10(5):1–8, 2004.

J. W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, Reading, MA, 1977.

J. W. Tukey. Robust techniques for the user. In Launer R.L. and Graham N.W., editors, *Robustness in Statistics*, pages 103–106, New York, 1979. Academic Press.

N. R. Ullman. The analysis of means (anom) for signal and noise. *Journal of Quality Technology*, 21:111–127, 1989.

G. N. Wilkinson. A general procedure for analysis of variance. *Biometrika*, 57:19–46, 1970.

F. Xhonga. Direct gold alloys - part II. *Journal of the American Academy of Gold Foil Operators*, 14:5–15, 1971.

G. Y. Yatracos. A note on Tukey's polyefficiency. *Biometrika*, 78(3):702–703, 1991.

Appendix A

A note by Dr. P. Craig on estimated minimum badness

We have data $B(\gamma_i)$ for $i = 1, \ldots, n$ near to the minimum of the expected badness. Assume the expected badness $b(\gamma)$ is quadratic near the minimum and that $B(\gamma_i)$ is an unbiased estimate of $b(\gamma_i)$ and that $\mathrm{Var}[\,\gamma_i\,]$ does not depend on i. For the moment we will also assume that $B(\gamma_1), \ldots, B(\gamma_n)$ are independent. We are interested in estimating $\min_\gamma b(\gamma)$ and we have two approaches: (i) $B_m = \min_i B(\gamma_i)$ and (ii) find i for which $B(\gamma_i)$ is minimum, write $\hat{\gamma}$ for that cut-off and then make new observation $B^*(\hat{\gamma})$. The purpose of what follows is to study the bias involved in the two methods. We believe B_m is mostly biased below and $B^*(\hat{\gamma})$ is biased above. The question is which bias is worse. Note that $B^*(\hat{\gamma})$ is an unbiased esimate of $b(\hat{\gamma})$ and so the bias is $\mathrm{E}[\,b(\hat{\gamma})\,]$.

As a metaphor for our real situation we will study the following problem. We observe independent $Y_0, \ldots, Y_m$ where $Y_i \sim (\mu_i, 1)$ for $\mu_i = \alpha(x_i - \frac{1}{2})^2$ and $x_i = i/m$.

The parameter α represents the ratio of the sampling standard deviation of $B(\gamma_i)$ to the difference between the maximum and minimum of $b(\gamma)$ on the range we are considering. m is effectively $n - 1$.

For even values of m, we have a best-case situation where we actually make an observation of badness at the minimum and for even values we have a worst-case situation where the minimum is in the middle between two of our observations.

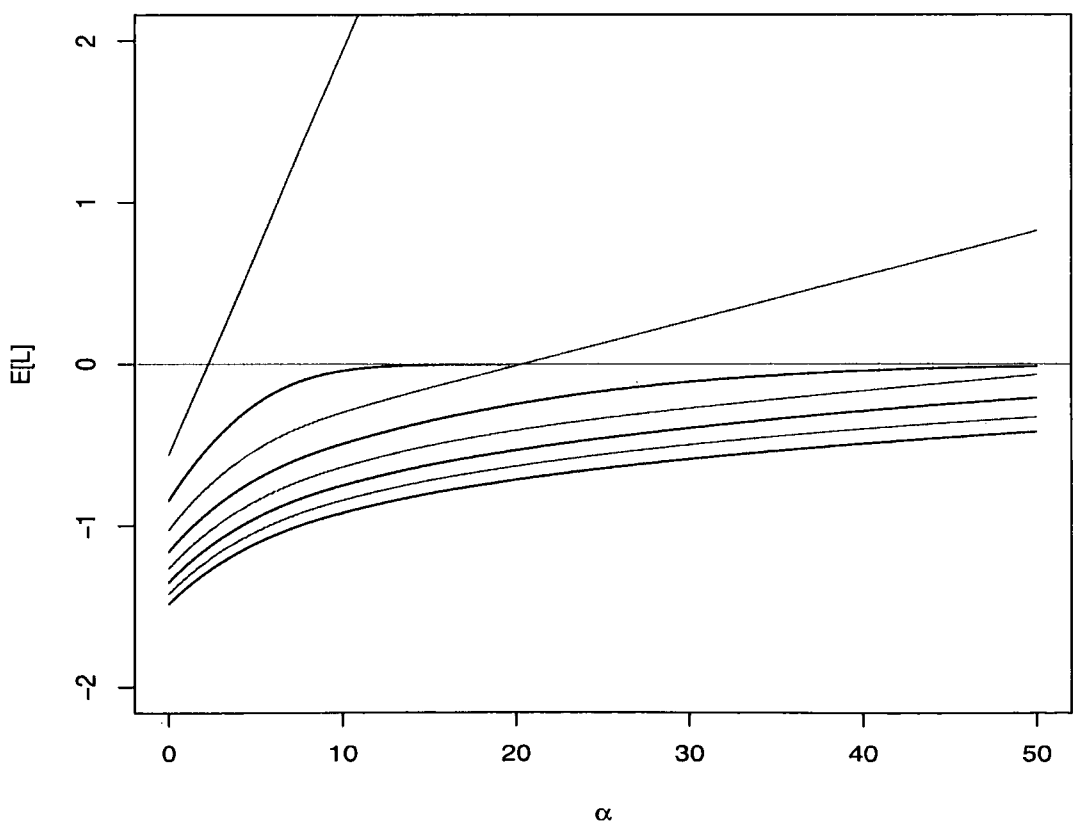
Let $L = \min_i Y_i$. The cdf of L is

$$F_L(y) = 1 - \prod_i [1 - \Phi(y - \mu_i)]$$

and we can approximate E[L] numerically in R using the `integrate` function based on the general formula for obtaining the expectation of a random quantity from the cdf:

$$\mathrm{E}[\,X\,] = \int_0^\infty [1 - F(x)]\,dx - \int_{-\infty}^0 F(x)\,dx$$

The results are shown in the picture below. The red curves are for $m = 1$, 3, 5 and 7 (m increasing as you go down the picture). The black curves are for $m = 2$, 4, 6, 8 (again m increasing as you go down the picture). The bias is potentially quite large for small α, i.e. when the badness curve is nearly flat compared to the sampling variation. If the the sampling errors are small relative to the change in badness from one cutoff to another, the bias is small. In either case the bias is relative to the sampling standard deviation.

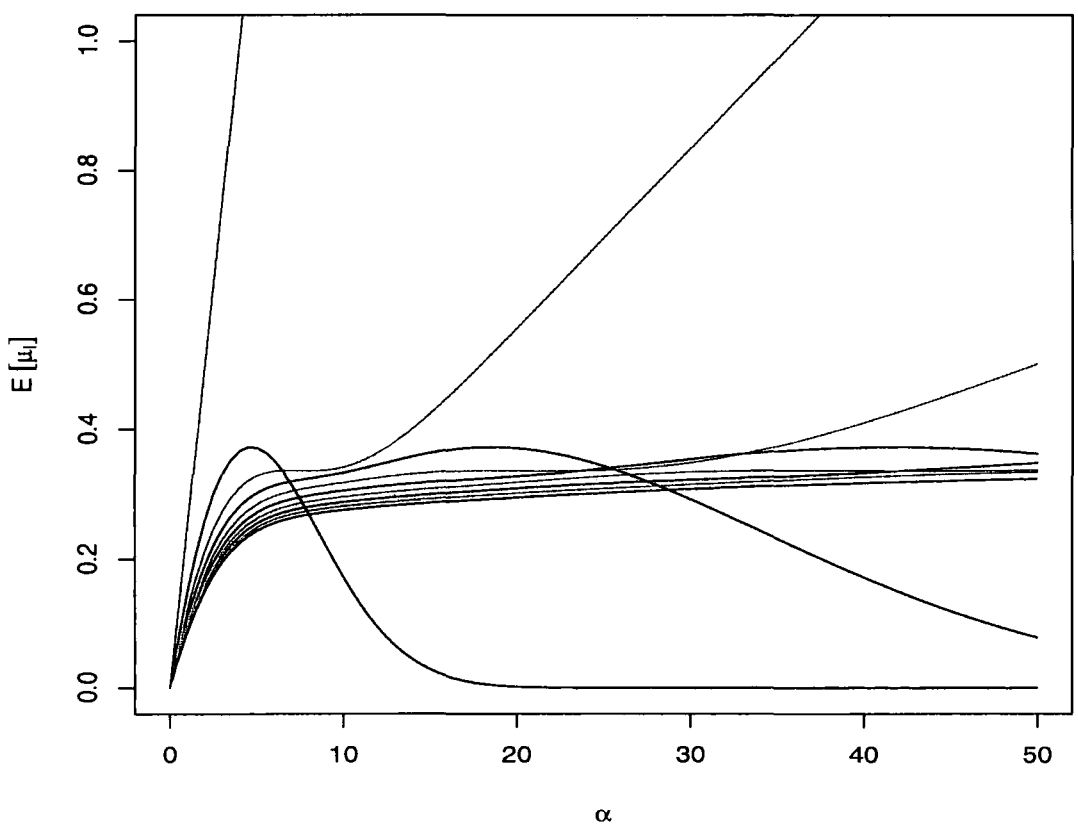


Now let I be the (random) value for which $Y_I = \min_j Y_j$. Then

$$P[I = i] = \int_{-\infty}^{\infty} \phi(y - \mu_i) \prod_{j \neq i} [1 - \Phi(y - \mu_j)]\, dy$$

which again may be approximated using `integrate`. We want to find $\mathrm{E}[\mu_I] = \mathrm{E}[\alpha(x_I - \frac{1}{2})^2]$ which is easy once we have computed the probability distribution of I.

The results are shown in the picture below. The red curves are for $m = 1$, 3, 5 and 7 (m increasing as you go down the picture). The black curves are for $m = 2$, 4, 6, 8 (again m increasing as you go down the picture at $\alpha = .5$). The bias is effectively bounded by .4 unless m is very small. On the other hand, the bias takes a long time to decay away for even m and becomes very large for odd m if α becomes very large. As before the bias is being measured relative to the sampling standard deviation. It would appear that this method is less likely to produce a bias which would cause problems in subsequent analyses.



Appendix B

R functions used in Chapter 2

B.1 bb0.sim

```
function (nn = 5:5, rr = 2000, ss = seq(0.7, 1.3, 0.05), kk = 1,
    p1 = "")
{
    for (jn in nn) {
        for (ii in 1:kk) {
            BBB = b.0way.0(matrix(rslash.0(rr * jn), jn, rr), ss)
            aa <- apply(BBB[[1]][, -1], 1, mean)
            dput(cbind(ss, aa), paste(p1, paste(jn, rr, kk, ii),
                sep = ""))
        }
    }
}
```

B.2 rslash.0

```
function (nnnn)
{
    rnorm(nnnn)/(runif(nnnn))
}
```

B.3 b.0way.0

```
function (aaaa, cfs)
{
    rsg <- aaaa
    kr = dim(aaaa)[2]
    n = dim(aaaa)[1]
    aaaa <- rbind(aaaa, rep(0, kr))
    aaaa <- apply(aaaa, 2, sweeper.0, swp.f = "lomedian")
    aaaa <- aaaa[1:n, ]
    oooo <- apply(abs(aaaa), 2, order, decreasing = T)
    rsg <- matrix(rsg[oooo + (col(oooo) - 1) * nrow(oooo)], n,
        kr)
    aaaa <- apply(abs(aaaa), 2, sort, decreasing = T)[1:(n -
        1), ]
    ii = 1:(n - 1)
    ref.vals <- qnorm(1 - (3 * ii - 1)/(6 * (n - 1) + 2))
    aaaa <- aaaa/ref.vals
    lambda <- t(t(aaaa)/apply(aaaa, 2, lomedian))
    lambda <- rbind(lambda, rep(0, kr))
    B = nB = c()
    for (icf in cfs) {
        ss = matrix(1, n, kr)
        ss[lambda <= icf] = 0
        ss = apply(ss, 2, cumprod)
        ss = 1 - ss
        nn <- apply(ss, 2, sum)
        bb = (apply(rsg * ss, 2, sum)/nn)^2
        B = rbind(B, c(icf, bb))
        nB = rbind(nB, c(icf, nn))
    }
    list(B, nB)
}
```

B.4 sweeper.0

```
function (vv, swp.f)
{
    ll = length(vv)
    tt = vv[1:ll - 1]
    vvv = tt[!is.na(tt)]
    lll = length(vvv)
    aaa <- switch(swp.f, mean = mean(vvv), median = median(vvv),
        lomedian = sort(vvv)[ceiling(lll/2)], fibian = {
            aba = median(vvv)
            if (floor(lll/2) * 2 == lll) {
                lom = sort(vvv)[lll/2]
                him = sort(vvv)[lll/2 + 1]
                if (abs(lom + vv[ll]) < abs(him + vv[ll]))
                  aba = lom
                else if (abs(lom + vv[ll]) > abs(him + vv[ll]))
                  aba = him
            }
            aba
        }, "")
    if (aaa == "")
        return("Error: Unknown sweeper function")
    vv[ll] = vv[ll] + aaa
    vv[1:ll - 1] = vv[1:ll - 1] - aaa
    vv
}
```

B.5 lomedian

```
function (vv)
{
    svv = sort(vv)
```

```
    svv[ceiling(length(svv)/2)]
}
```

B.6 bb0.sim.res

```
function (p1 = "arc/B0.slash/1/", ns = c(4:30, 49, 50), ll = 50,
    rr = 2e+05)
{
    res = c()
    for (nn in ns) {
        kk = seq(0.7, 1.3, 0.01)
        for (i in 1:ll) kk = cbind(kk, dget(paste(p1, nn, " ",
            rr, " ", ll, " ", i, sep = ""))[, 2])
        kk = cbind(kk, t(apply(kk[, -1], 1, ssummary)))
        res = rbind(res, c(nn, kk[, -c(2:(ll + 1))][kk[, ll +
            2] == min(kk[, ll + 2])], ))
    }
    colnames(res) = c("n", "cut-off/minB", "min B", "stdev",
        "C.I. %95 L.", "C.I. %95 U.")
    res
}
```

B.7 pp0.sim

```
function (r = 200, n = 5, cfs = seq(0.8, 1.6, 0.05), cp = 1)
{
    slash = matrix(rnorm(r * n)/runif(r * n), n, r)
    gauss.conv = qnorm(pnorm(slash) + (dnorm(slash) - dnorm(0))/slash)
    BBB = b.0way.0(cbind(gauss.conv, slash), cfs)
    BmB = matrix(apply(rbind(BBB[[1]][, -1][, 1:r], BBB[[1]][,
        -1][, (r + 1):(2 * r)]), 1, mean), length(cfs), 2)
    BvB = matrix(apply(rbind(BBB[[1]][, -1][, 1:r], BBB[[1]][,
        -1][, (r + 1):(2 * r)]), 1, var), length(cfs), 2)
```

```
    colnames(BmB) = colnames(BvB) = c("Normal", "Slash")
    BmB[, 1] = BmB[, 1]/(1/n)
    BmB[, 2] = BmB[, 2]/bb0simres[, 3][bb0simres[, 1] == n]
    dput(list(scaled.mean = cbind(cfs, BmB), unscaled.variance = cbind(cfs,
        BvB), remained.obs.n = cbind(cfs = c(normal = cfs, slash = cfs),
        t(apply(rbind(BBB[[2]][, -1][, 1:r], BBB[[2]][, -1][,
            (r + 1):(2 * r)]), 1, function(x) c(min = min(x),
            Max = max(x), Mean = mean(x))))), c(n = n, r = r,
        cp = cp)), paste(n, r, cp))
}
```

B.8 pp0.sim.res

```
function (ns = 5, r = 1e+06, cps = 2:4, p1 = "~/arc/B0.sn/",
    p2 = "", log = "", plpl = 0)
{
    BB = xlxl = ylyl = c()
    for (n in ns) {
        for (cp in cps) {
            BmB = dget(paste(p1, n, " ", r, " ", cp, p2, sep = ""))$scaled.mean
            xlxl = summary(c(xlxl, BmB[, 1]))[c(1, 6)]
            ylyl = summary(c(ylyl, as.vector(BmB[, -1])))[c(1,
                6)]
            BB = cbind(BB, BmB)
        }
    }
    if (plpl == 0) {
        ccp = 0
        for (n in ns) {
            for (cp in cps) {
                plot(BB[, ccp + 1], BB[, ccp + 2], pch = "N",
                  "o", lwd = 1, ylim = ylyl, xlim = xlxl, lty = cp,
                  main = paste("n=", n, ", r=", r, "(", paste(cps,
```

```
                collapse = ","), ")"), xlab = "Cut-off values",
              ylab = "Scaled Badness", col = "green", log = log)
            par(new = T)
            plot(BB[, ccp + 1], BB[, ccp + 3], pch = as.character(cp),
              "o", lwd = 2, ylim = ylyl, xlim = xlxl, lty = cp,
              xlab = "", ylab = "", col = "red", log = log)
            par(new = T)
            ccp = ccp + 3
        }
    }
    par(new = F)
    BB
}
else if (prod(apply(BB[, (1:length(cps)) * 3 - 2], 1, var) ==
    0) == 1) {
    BBm = cbind(BB[, 1], apply(BB[, (1:length(cps)) * 3 -
        1], 1, mean), apply(BB[, (1:length(cps)) * 3], 1,
        mean))
    plot(BB[, 1], apply(BB[, (1:length(cps)) * 3 - 1], 1,
        mean), ylim = ylyl, "o", pch = "N", main = paste("n=",
        n, ",", r, "*", length(cps), "(", paste(cps, collapse = ","),
        ")"), xlab = paste("Cut-off values", "- intersection at:",
        round(mlines.int(BBm)$x, 4)), ylab = "Scaled Badness",
        col = "green")
    par(new = T)
    c(n, round(mlines.int(BBm)$x, 4))
    plot(BB[, 1], apply(BB[, (1:length(cps)) * 3], 1, mean),
        ylim = ylyl, "o", pch = "S", lty = 2, xlab = "",
        ylab = "", col = "red")
    c(n, round(mlines.int(BBm)$x, 4))
}
else {
    kk = "Error: Cut-off sets are not the same!"
```

```
        kk
    }
}
```

B.9 b.0way

```
function (r = 200, n = 5, cfs = seq(0.7, 1.6, 0.05), cp = 1,
    method = "ST", swpf = "lomedian", sclf = "lomedian", sqlm = F)
{
    slash = matrix(rnorm(r * n)/runif(r * n), n, r)
    gauss.conv = qnorm(pnorm(slash) + (dnorm(slash) - dnorm(0))/slash)
    BBB = switch(method, ST = b.0way.ST(cbind(gauss.conv, slash),
        cfs, swpf, sclf, sqlm), MZ = b.0way.MZ(cbind(gauss.conv,
        slash), cfs))
    BmB = matrix(apply(rbind(BBB[[1]][, -1][, 1:r], BBB[[1]][,
        -1][, (r + 1):(2 * r)]), 1, mean), length(cfs), 2)
    BvB = matrix(apply(rbind(BBB[[1]][, -1][, 1:r], BBB[[1]][,
        -1][, (r + 1):(2 * r)]), 1, var), length(cfs), 2)
    colnames(BmB) = colnames(BvB) = c("Normal", "Slash")
    list(unscaled.mean = cbind(cfs, BmB), unscaled.variance = cbind(cfs,
        BvB), remained.obs.n = cbind(cfs = c(normal = cfs, slash = cfs),
        t(apply(rbind(BBB[[2]][, -1][, 1:r], BBB[[2]][, -1][,
            (r + 1):(2 * r)]), 1, function(x) c(min = min(x),
            Max = max(x), Mean = mean(x))))), c(n = n, r = r,
        cp = cp, method = method))
}
```

B.10 b.0way.ST

```
function (aaaa, cfs, swpf = "lomedian", sclf = "lomedian")
{
    rsg <- aaaa
    kr = dim(aaaa)[2]
```

```
    n = dim(aaaa)[1]
    aaaa <- rbind(aaaa, rep(0, kr))
    aaaa <- apply(aaaa, 2, sweeper.0, swp.f = swpf)
    aaaa <- aaaa[1:n, ]
    oooo <- apply(abs(aaaa), 2, order, decreasing = T)
    rsg <- matrix(rsg[oooo + (col(oooo) - 1) * nrow(oooo)], n,
        kr)
    aaaa <- apply(abs(aaaa), 2, sort, decreasing = T)[1:(n -
        1), ]
    ii = 1:(n - 1)
    ref.vals <- qnorm(1 - (3 * ii - 1)/(6 * (n - 1) + 4))
    aaaa <- aaaa/ref.vals
    lambda <- t(t(aaaa)/apply(aaaa, 2, sclf))
    lambda <- rbind(lambda, rep(0, kr))
    B = nB = c()
    for (icf in cfs) {
        ss = matrix(1, n, kr)
        ss[lambda <= round(icf, 2)] = 0
        ss = apply(ss, 2, cumprod)
        ss = 1 - ss
        nn <- apply(ss, 2, sum)
        bb = (apply(rsg * ss, 2, sum)/nn)^2
        B = rbind(B, c(icf, bb))
        nB = rbind(nB, c(icf, nn))
    }
    list(B, nB)
}
```

B.11 b.0way.MZ

```
function (aaaa, cfs)
{
    rsg <- aaaa
```

```
    kr = dim(aaaa)[2]
    n = dim(aaaa)[1]
    lambda = apply(aaaa, 2, function(zz) abs(0.6745 * (zz - median(zz))/
        median(abs(zz - median(zz)))))
    B = nB = c()
    for (icf in cfs) {
        ss = matrix(1, n, kr)
        ss[lambda > icf] = 0
        nn <- apply(ss, 2, sum)
        bb = (apply(rsg * ss, 2, sum)/nn)^2
        B = rbind(B, c(icf, bb))
        nB = rbind(nB, c(icf, nn))
    }
    list(B, nB)
}
```

B.12 b.0way.res.6.ci

```
function (nn = 15, method = "ST", location = "arc/res.short-cfs/",
    res.table = b.0way.res.table.1, kk = 2)
{
    fff = ccc(1, nn, method, location, res.table)[[1]][, 1]
    lll = length(fff)
    ddd = c()
    for (j in 1:(lll * kk)) ddd = cbind(ddd, ccc(j, nn, method,
        location, res.table)[[1]][, 2])
    LU = cir(t(ddd)[cbind(1:(lll * kk), rep(1:lll, kk))], rep(fff,
        kk))
    par(mfrow = c(2, 2))
    plot(lm(t(ddd)[cbind(1:(lll * kk), rep(1:lll, kk))] ~ rep(fff,
        kk)), main = paste(method, nn))
    list(lm(t(ddd)[cbind(1:(lll * kk), rep(1:lll, kk))] ~ rep(fff,
        kk) + I(rep(fff, kk)^2)), LU, lm(t(ddd)[cbind(1:(lll *
```

```
        kk), rep(1:lll, kk))] ~ rep(fff, kk)))
}
```

B.13 b.0way.summ

```
function (nnnn = c(4:20, 30), rep1 = 10000, rep2 = 3)
{
    aaaa = c()
    for (nn in rep(nnnn, rep2)) {
        cccc <- apply(apply(matrix(rslash(rep1 * nn), rep1, nn),
            1, function(xxxx) {
                STm1(xxxx, cu = b.0way.res.table[b.0way.res.table[,
                  1] == nn, 7])$r
            }), 2, sum)
        bbbb <- c(nn, mean(cccc/nn), table(cccc))
        aaaa <- c(aaaa, bbbb)
    }
    aaaa
}
```

B.14 STm1

```
function (xxx, swp = "lomedian", scalf = "lomedian", cutoff = 1.5)
{
    if (length(dim(xxx)) == 0)
        ddff = length(xxx) - 1
    else ddff = prod(dim(xxx) - 1)
    xx = as.vector(xxx)
    aa2 = xx - switch(swp, lomedian = lomedian(xx), median = median(xx))
    oo3 = order(abs(aa2))
    aa3 = sort(abs(aa2))
    nnzz <- sum(aa3 != 0)
    if (nnzz < ddff)
```

```
        nnn = min(ddff, sum(aa2 != 0) + 1)
    else {
        aa3 = aa3 - aa3[length(xxx):1][ddff + 1]
        nnn = ddff
    }
    aa4 = c(rep(NA, length(xx) - nnn), qnorm(((nnn - (nnn:1) +
        1)/(nnn + 2/3) + 1)/2))
    aa5 = aa3/aa4
    aa6 = aa5/switch(scalf, lomedian = lomedian(aa5), median = median(aa5,
        na.rm = T))
    aa7 = rep(1e+06, length(xx))
    aa7[oo3] = cumprod(((aa6 > cutoff)^2)[length(xx):1])[length(xx):1]
    aa7[is.na(aa7)] = 0
    list(details = cbind(xx, aa2, oo3, aa3, aa4, aa5, aa6, aa7),
        resault = array(aa7, dim(as.array(xxx))))
}
```

Appendix C

R functions used in Chapter 3

C.1 b.2way.new.2

```
function (lvls = c(5, 4), mdl = NULL, data = NULL, rr = 8, cfs = seq(0.7,
    3, 0.1), swpf = NE.median, sclf = "lomedian", effecttoadd = c(1,
    2), sqlm = F)
{
    if (is.null(mdl)) {
        fr = factor(rep(1:lvls[1], rep(lvls[2], lvls[1])))
        fc = factor(rep(1:lvls[2], prod(lvls)/lvls[2]))
        data = data.frame(fr, fc)
        mdl = ~fr + fc
    }
    des.m = incidence.matrix.2(mdl, data)
    dims <- table(des.m[[3]][des.m[[2]] == 1])
    rrow = dims[1]
    ccol = dims[2]
    slash = matrix(rslash(nrow(des.m[[1]]) * rr, C = T, M = 80),
        nrow(des.m[[1]]), rr)
    gauss.conv = qnorm((pslash(slash) - pslash(-80))/(pslash(80) -
        pslash(-80)))
    if (effecttoadd[1] != 0) {
        cc2 = matrix(0, sum(dims) + 1, rr)
```

```
    for (jj in 1:length(effecttoadd)) {
        cc = matrix(rnorm(dims[effecttoadd[jj]] * rr), ncol = rr)
        cc2[des.m[[3]] == effecttoadd[jj], ] <- matrix(as.vector(t(cc)) -
            c(rep(1, nrow(cc)) %*% cc/nrow(cc)), ncol = rr,
            byrow = T)
    }
    cc3 <- des.m[[1]] %*% cc2
    slash <- cc3 + slash
    gauss.conv <- cc3 + gauss.conv
}
rrnd = cbind(gauss.conv, slash)
rres = apply(t(t(rrnd) - apply(rrnd, 2, median)), 2, function(xxxx) {
    as.vector(n0gg.avg.2way(n2gg(matrix(xxxx, ccol, rrow)),
        swp = "NE.median")[-(ccol + 1), -(rrow + 1)])
})
BBad = c()
icfs = cfs[length(cfs):1]
for (cf in icfs) {
    rnd2 = rrnd
    iind = apply(rres, 2, function(xxxx) {
        STm2(xxxx, ddff = (rrow - 1) * (ccol - 1), cutoff = cf,
            scalf = sclf)$r
    })
    if (sum(apply(iind == 0, 2, function(xxxx) {
        mat.rank(des.m[[1]][xxxx, ])
    }) - (sum(lvls) - 1)) != 0)
        break
    rnd2[iind == 1] = NA
    A = apply(rnd2, 2, function(xxxx) {
        yyyy <- n0gg(n2gg(matrix(xxxx, rrow, ccol, byrow = T)),
            swp = "mean")
        c(row.coef = yyyy[-(rrow + 1), (ccol + 1)], col.coef = yyyy[(rrow +
            1), -(ccol + 1)])
```

```
        })
        if (effecttoadd[1] != 0)
            A <- A - cbind(cc2[-1, ], cc2[-1, ])
        br <- apply(A[des.m[[3]][-1] == 1, ]^2, 2, sum)
        bc <- apply(A[des.m[[3]][-1] == 2, ]^2, 2, sum)
        Bads = rbind(br, bc)
        Bads = rbind(Bads, ccol * Bads[1, ] + rrow * Bads[2,
            ])
        BBad = rbind(BBad, cbind(c(cf, cf, cf), Bads))
    }
    if (length(BBad) == 0)
        break
    if (length(cfs) > 1 & cf >= 1)
        dput(list(lvls, rrnd, iind), paste("rank", date()))
    if (sqlm) {
        nnn = rrow * ccol
        half.gq = sort(qnorm(((nnn - (nnn:1) + 1)/(nnn + 2/3) +
            1)/2))
        SqLm = apply(apply(rrnd, 2, function(xxxx) sort(abs(xxxx)))/half.gq,
            2, lomedian)^2
        BBad[, -1] = t(t(BBad[, -1])/SqLm)
    }
    BBad = cbind(BBad[, 1], BBad[, 2:(rr + 1)] %*% rep(1, rr)/rr,
        BBad[, (rr + 2):(rr * 2 + 1)] %*% rep(1, rr)/rr, BBad[,
            2:(rr + 1)]^2 %*% rep(1, rr)/rr, BBad[, (rr + 2):(rr *
            2 + 1)]^2 %*% rep(1, rr)/rr)
    dimnames(BBad) = list(rep(c("Br", "Bc", "BT"), dim(BBad)[1]/3),
        c("cfs", "Mean.Gauss", "Mean.Slash", "MeanSq.Gauss",
            "MeanSq.Slash"))
    list(BBad, rr = rr)
}
```

C.2 incidence.matrix.2

```
function (mdl, dt = NULL)
{
    options(contrasts = c("contr.treatment", "contr.poly"))
    ddtt = ""
    if (!is.null(dt))
        ddtt = "dt$"
    tt <- attr(terms(mdl, data = dt), "term.labels")
    oo <- attr(terms(mdl, data = dt), "order")
    ff = ee = dd = c()
    for (ii in 1:length(tt)) {
        if (!is.null(dt))
            aa <- eval(parse(text = paste("model.matrix(~", tt[ii],
                ",data=dt)")))
        else aa <- eval(parse(text = paste("model.matrix(~",
            tt[ii], ")")))
        bb <- aa[, 1] - apply(as.matrix(aa[, -1]), 1, sum)
        if (sum(bb) == 0)
            aa <- aa[, -1]
        else {
            aa[, 1] <- bb
            dimnames(aa)[[2]][1] = paste(tt[ii], eval(parse(text = paste("levels(",
                ddtt, tt[ii], ")")))[1], sep = "")
        }
        dd = cbind(dd, aa)
        ee = c(ee, rep(oo[ii], dim(aa)[2]))
        ff = c(ff, rep(ii, dim(aa)[2]))
    }
    list(X = cbind(Intercept = 1, dd), lvls = cbind(lvls1 = c(0,
        ee), lvls2 = c(0, ff)), neffects = c("Intercept", tt))
}
```

C.3 rslash

```
function (nnnn, Censoring = F, M = 70)
{
    if (!Censoring)
        rnorm(nnnn)/(runif(nnnn))
    else {
        a = rnorm(nnnn * 1.05)/(runif(nnnn * 1.05))
        a[abs(a) < M][1:nnnn]
    }
}
```

C.4 pslash

```
function (s)
{
    t <- pnorm(s) + (dnorm(s) - dnorm(0))/s
    t[abs(s) < 1e-07] = 0.5
    t
}
```

C.5 n0gg.avg.2way

```
function (aa, swp)
{
    bb = c()
    while (!isTRUE(all.equal(aa, bb))) {
        bb = aa
        aa <- (n0gg(bb, swp, st = "first", rep = 1) + n0gg(bb,
            swp, st = "last", rep = 1))/2
    }
    aa
}
```

C.6 n0gg

```
{
    for (l in 1:rep) {
        bb = aa
        for (i in switch(starting.direction, first = 1:length(dim(aa)),
            last = length(dim(aa)):1)) aa = n1gg(aa, i, swp)
        if (isTRUE(all.equal(bb, aa)))
            break
    }
    aa
}
```

C.7 n1gg

```
function (aa, ppmm, swp.f)
{
    iiii = 1:length(dim(aa))
    bbbb = iiii[ppmm]
    iiii[ppmm] = iiii[1]
    iiii[1] = bbbb
    aa = aperm(aa, iiii)
    dd = dim(aa)
    d1 = dd[1]
    d2 = prod(dd)/dd[1]
    aa = array(aa, c(d1, d2))
    for (i in 1:d2) {
        aa[, i] = sweeper(aa[, i], swp.f)
    }
    aa = array(aa, dd)
    aa = aperm(aa, iiii)
    aa
}
```

C.8 sweeper

```
function (vv, swp.f)
{
    ll = length(vv)
    tt = vv[1:ll - 1]
    vvv = tt[!is.na(tt)]
    lll = length(vvv)
    aaa <- switch(swp.f, mean = mean(vvv), median = median(vvv),
        lomedian = lomedian(vvv), himedian = himedian(vvv),
        NE.median = NE.median(vvv),
        fibian = {
            aba = median(vvv)
            if (floor(lll/2) * 2 == lll) {
                lom = sort(vvv)[lll/2]
                him = sort(vvv)[lll/2 + 1]
                if (abs(lom + vv[ll]) < abs(him + vv[ll]))
                  aba = lom
                else if (abs(lom + vv[ll]) > abs(him + vv[ll]))
                  aba = him
            }
            aba
        }, "")
    if (aaa == "")
        return("Error: Unknown sweeper function")
    vv[ll] = vv[ll] + aaa
    vv[1:ll - 1] = vv[1:ll - 1] - aaa
    vv
}
```

C.9 himedian

```
function (vv)
```

```
{
    svv = sort(vv)
    svv[floor(length(svv)/2) + 1]
}
```

C.10 NE.median

```
function (vv)
{
    nemed = lomedian(vv)
    if (abs(himedian(vv)) < abs(lomedian(vv)))
        nemed = himedian(vv)
    nemed
}
```

C.11 n2gg

```
function (aa)
{
    dd = dim(aa)
    bb = as.vector(aa)
    ddd = 1
    ggg = 0
    fff = 1
    for (i in 1:length(dd)) {
        ddd = ddd * dd[i]
        ggg = ggg + fff
        fff = ggg * dd[i]
        cc = c()
        for (j in 1:(prod(dd)/ddd)) {
            cc = c(cc, bb[((j - 1) * fff + 1):(j * fff)], rep(0,
                ggg))
        }
```

```
        bb = cc
    }
    array(bb, dd + 1)
}
```

C.12 STm2

```
function (xx, scalf = "lomedian", cutoff = 1.5, ddff)
{
    oo3 = order(abs(xx), decreasing = TRUE)
    aa3 = sort(abs(xx), decreasing = TRUE)
    nnzz = sum(xx != 0)
    nndf = ddff
    if (nnzz < ddff)
        nndf = min(ddff, sum(xx != 0) + 1)
    else if (nnzz > ddff)
        aa3 = aa3 - aa3[ddff + 1]
    aa4 = c(qnorm(((nndf - (1:nndf) + 1)/(nndf + 2/3) + 1)/2),
        rep(NA, length(xx) - nndf))
    aa5 = aa3/aa4
    aa6 = aa5/switch(scalf, lomedian = lomedian(aa5), himedian = himedian(aa5),
        median = median(aa5, na.rm = TRUE))
    aa7 = rep(1e+06, length(xx))
    aa7[oo3] = cumprod(((aa6 > cutoff)^2))
    aa7[is.na(aa7)] = 0
    list(details = cbind(xx, oo3, aa3, aa4, aa5, aa6, aa7), result = aa7)
}
```

C.13 mat.rank

```
function (xxxx)
{
    aaaa = svd(xxxx, nu = 0, nv = 0)
```

```
    sum(aaaa$d > 1e-04 * aaaa$d[1])
}
```

C.14 b.2way.new.2.res

```
function (lvls = c(5, 4), swpf = "NE.median", sclf = "lomedian",
    rr = 1:10, location = "~/arc/final-now/", start.order = "",
    effecttoadd = c(1, 2), rept = 10000, prts = F, whichB = c("Br",
        "Bc", "BT"), comma = ",", cfseq = "0.1.2.7.0.1.")
{
    pt = function(a, lcfs) {
        iter = length(a)/lcfs/5
        if (floor(iter) != iter) {
            dput(a, "a")
            return(paste("########################", paste(lvls,
                collapse = ""), "############################"))
        }
        a <- apply(array(a, c(lcfs, 5, iter)), c(1, 2), mean)
        row.names(a) <- row.names(aa[[1]])
        par(mfrow = c(1, length(whichB)))
        cc = c()
        for (jj in whichB) {
            b <- a[row.names(a) == jj, ]
            minBg = switch(jj, Bc = (lvls[2] - 1)/lvls[1], Br = (lvls[1] -
                1)/lvls[2], BT = lvls[1] + lvls[2] - 2)
            minBs = min(b[, 3])
            b = cbind(b, sqrt((b[, 5] - b[, 3]^2) * iter * rept/(iter *
                rept - 1)))
            plot(rep(b[, 1], 2), c(b[, 2]/minBg, b[, 3]/minBs),
                ylab = "", xlab = "", pch = rep(c("o", "*"),
                  rep(dim(b)[1], 2)))
            legend("topleft", c("Gaussian", "Slash"), pch = c("o",
                "*"))
```

```
        cutpoint = mlines.int(cbind(b[, 1], b[, 2]/minBg,
            b[, 3]/minBs))
        gg = b[b[, 2] == min(b[, 2]), 1:2]
        if (is.matrix(gg)) {
            print(paste("Cut-off's", paste(gg[, 1], collapse = ","),
              "have the same (minimum) badness in Gaussian"))
            gg = gg[1, ]
        }
        ss = b[b[, 3] == min(b[, 3]), c(1, 6, 3)]
        if (is.matrix(ss)) {
            print(paste("Cut-off's", paste(ss[, 1], collapse = ","),
              "have the same (minimum) badness in Slash"))
            ss = ss[1, ]
        }
        cc = c(c(gg, ss, cutpoint$x, ), cc)
        names(cc)[1:6] = c(paste(jj, c("minG.cf", "minG.nu",
            "minBs.cf", "minS SD", "minS", "cf-mm", )))
        title(main = paste(lvls[1], "x", lvls[2], "-", swpf,
            "-", start.order, "-", effecttoadd[1], ",", effecttoadd[2],
            "-", sclf, "-", aa[[2]], "(", switch(as.character(iter),
              "1" = ii, ind), ")", sep = ""), jj, xlab = "cut-off values",
            sub = paste("minB(Nu.)=", round(min(b[, 2]),
              5), ".. minimax-cf=", round(cutpoint$x, 4)))
    }
    cc
}
summ = ptr = c()
a = ind = c()
for (ii in rr) {
    aa = dget(paste(location, lvls[1], comma, lvls[2], ".",
        swpf, ".", start.order, effecttoadd[1], comma, effecttoadd[2],
        ".", rept, ".", cfseq, ii, sep = ""))
    a = c(a, aa[[1]])
```

```
        lcfs <- dim(aa[[1]])[1]
        if (!is.null(aa))
            ind = paste(c(ind, ii), collapse = "")
        ptr = rbind(ptr, c(ii, pt(aa[[1]], lcfs)))
        if (prts == T)
            dev.print()
    }
    for (jj in whichB) summ = cbind(summ, apply(ptr[, grep(jj,
        dimnames(ptr)[[2]])], 2, function(yy) {
        xx = as.numeric(yy)
        c(mean = mean(xx), var = var(xx), cv = sqrt(var(xx))/mean(xx))
    }))
    summ = rbind(summ, overall = pt(a, lcfs))
    list(header = c(lvls[1], lvls[2], swpf, sclf, rept, cfseq),
        details = ptr, summary = summ)
}
```

C.15 b.2way.new.2.res.m2nd

```
function (cfseq = "0.1.2.7.0.1.")
{
    tt = c()
    sink("qq")
    for (ii in 2:7) for (jj in ii:7) {
        a = b.2way.new.2.res(c(ii, jj), rr = c(1:10), rept = 10000,
            location = "~/arc/final-now/", start = "", comma = ",",
            cfseq = cfseq, w = "BT")
        b = b.2way.new.2.res(c(jj, ii), rr = c(1:10), rept = 10000,
            location = "~/arc/final-now/", start = "", comma = ",",
            cfseq = cfseq, w = "BT")
        if (a$s[4, 3] != b$s[4, 3])
            print(paste("Warning: gamma_m in(", ii, ",", jj,
                "is not the same as(", jj, ",", ii, ")"))
```

```
        print(paste("cdr-awk3", paste(ii, jj, sep = ","), "NULL NULL 10000",
            a$s[4, 3], a$s[4, 3], "0 NE.median lomedian 1,2 10"))
        for (kk in seq(round(mean(a$s[4, 6], b$s[4, 6]) - 0.05,
            2), by = 0.01, length = 11)) print(paste("cdr-awk3",
            paste(ii, jj, sep = ","), "NULL NULL 10000", sprintf("%.2f",
                kk), sprintf("%.2f", kk), "0 NE.median lomedian 1,2 2"))
        tt = rbind(tt, c(ii, jj, a$s[4, ]))
        if (ii != jj)
            tt = rbind(tt, c(jj, ii, b$s[4, ]))
    }
    sink()
    colnames(tt)[1] = cfseq
    dput(tt, "b.2way.new.2.res.tbl")
}
```

C.16 b.2way.new.2.res.mf

```
function (lvls = c(7, 7), swpf = "NE.median", sclf = "lomedian",
    rr = 1:10, location = "~/arc/final-now/", ccff = 0, mmin1 = "th",
    title = T, start.order = "", effecttoadd = c(1, 2), rept = 10000,
    reps = T, prts = F, whichB = "BT", table = F, comma = ",")
{
    ttt = c()
    tt = dget("b.2way.new.2.res.tbl")
    kk = 1
    while (kk <= dim(tt)[1]) {
        overall <- tt[kk, ]
        cf = overall[5]
        lvls = overall[1:2]
        if (sum(lvls == 2) != 0) {
            ttt = rbind(ttt, rep(0, 5))
            kk = kk + 1
            next
```

```
        }
        b = c()
        for (ii in rr) {
            b = rbind(b, dget(paste(location, lvls[1], comma,
                lvls[2], ".", swpf, ".", start.order, effecttoadd[1],
                comma, effecttoadd[2], ".", rept, ".", cf, ".",
                cf, ".0.", ii, sep = ""))[[1]])
        }
        bb = b[row.names(b) == whichB, ]
        ttt = rbind(ttt, apply(bb, 2, mean))
        if (lvls[1] != lvls[2]) {
            ttt = rbind(ttt, 0)
            kk = kk + 1
        }
        kk = kk + 1
    }
    print(ttt)
    cbind(tt, ttt)
}
```

C.17 b.2way.new.2.res.ci

```
function (lvls = c(7, 7), swpf = "NE.median", sclf = "lomedian",
    rr = 1:22, location = "~/arc/final-now/", ccff = 0, mmin1 = "th",
    title = T, start.order = "", effecttoadd = c(1, 2), rept = 10000,
    reps = T, prts = F, whichB = "BT", table = F, comma = ",",
    cfs)
{
    tt = dget("b.2way.new.2.res.tbl")
    overall <- tt[tt[, 1] == lvls[1] & tt[, 2] == lvls[2], ]
    cfs = seq(round(overall[8] - 0.05, 2), by = 0.01, len = 11)
    b = c()
    for (cf in sprintf("%.2f", cfs)) for (ii in 1:2) {
```

```
    aa = dget(paste(location, lvls[1], comma, lvls[2], ".",
        swpf, ".", start.order, effecttoadd[1], comma, effecttoadd[2],
        ".", rept, ".", cf, ".", cf, ".0.", ii, sep = ""))
    if (is.null(aa))
        return(paste("Error in file:", paste(location, lvls[1],
            comma, lvls[2], ".", swpf, ".", start.order,
            effecttoadd[1], comma, effecttoadd[2], ".", rept,
            ".", cf, ".", cf, ".0.", ii, sep = "")))
    b = rbind(b, aa[[1]])
}
bb = b[row.names(b) == whichB, ]
cfs = as.numeric(names(table(bb[, 1])))
lcfs = length(cfs)
minBg = switch(whichB, Bc = (lvls[2] - 1)/lvls[1], Br = (lvls[1] -
    1)/lvls[2], BT = lvls[1] + lvls[2] - 2)
minBs = overall[7]
n = length(bb[, 1])
X = cbind(rep(1, n), bb[, 1])
Y = bb[, 2]/minBg - bb[, 3]/minBs
par(mfrow = c(3, 2))
plot(X[, 2], Y)
title(main = paste(lvls, collapse = "x"))
B = solve(t(X) %*% X) %*% (t(X) %*% Y)
E = Y - X %*% B
plot(lm(Y ~ X[, 2]))
S = sqrt(t(E) %*% E/(n - 2))
C = c(-1/B[2], B[1]/B[2]^2)
L1 = t(C) %*% B - qt(0.975, n - 2) * S * sqrt(t(C) %*% solve(t(X) %*%
    X) %*% C)
U1 = t(C) %*% B + qt(0.975, n - 2) * S * sqrt(t(C) %*% solve(t(X) %*%
    X) %*% C)
cc = c(0, as.numeric(paste(lvls, collapse = "")), min(bb[,
    2]), minBs, overall[8], -B[1]/B[2], c(L1, U1) - B[1]/B[2],
```

```
        shapiro.test(E)$p)
    cd <- lm(bb[, 3]/overall[4]^2 * overall[3]/sqrt(1e+05) ~
        bb[, 1])$coef
    VVT = matrix(c(cd[1]^2, prod(cd), prod(cd), cd[2]^2), 2,
        2)
    VarB = as.numeric(S)^2 * solve(t(X) %*% X) + VVT
    L2 = t(C) %*% B - qnorm(0.975) * sqrt(t(C) %*% VarB %*% C)
    U2 = t(C) %*% B + qnorm(0.975) * sqrt(t(C) %*% VarB %*% C)
    cc = rbind(cc, c(1, as.numeric(paste(lvls, collapse = "")),
        min(bb[, 2]), minBs, overall[5], -B[1]/B[2], c(L2, U2) -
            B[1]/B[2], shapiro.test(E)$p))
    colnames(cc) = c("Uncert.Ind.", "lvls", paste(whichB, c("minG.nu",
        "minS", "cf-mm", "cf-lm", "ci-lm-L", "ci-lm-U")), "Shapiro.p")
    cc
}
```

Appendix D

R function for the new algorithm of decomposition in chapter 4

Any called function which is not presented here, can be found in previous appendices.

D.1 decomp

```
function (mdl, dt = NULL, swp.f = mean, ico = NULL)
{
    options(contrasts = c("contr.treatment", "contr.poly"))
    aa = incidence.matrix(mdl, dt)
    if (!is.null(dt))
        y = eval(parse(text = paste("dt$", as.character(mdl)[2])))
    else y = eval(parse(text = paste(as.character(mdl)[2])))
    yna <- is.na(y)
    aa[[1]] = aa[[1]][!is.na(y), ]
    aa[[2]] = aa[[2]][apply(aa[[1]], 2, sum) != 0, ]
    aa[[1]] = aa[[1]][, apply(aa[[1]], 2, sum) != 0]
    y = y[!is.na(y)]
    ly = length(y)
    X = cbind(aa[[1]], diag(1, ly, ly))
    lvls1 = c(aa[[2]][, 1], rep(max(aa[[2]][, 1]) + 1, ly))
    lvls2 = c(aa[[2]][, 2], rep(max(aa[[2]][, 2]) + 1, ly))
```

```
co <- rep(0, ncol(X))
iiend <- lvls1 == max(lvls1)
if (identical(swp.f, NE.median)) {
    med.y = median(y, na.rm = TRUE)
    y = y - med.y
}
co[iiend] = apply(X[, iiend] * y, 2, sum, na.rm = TRUE)/apply(X[,
    iiend] != 0, 2, sum)
if (!is.null(ico))
    co = ico
pivot = function(X, coe, iiis, iis) {
    ipivot = function(X, coef, iiiss, iis) {
        for (ii in (1:length(iiiss))[iiiss]) {
            if (!is.child(colnames(X)[iis], colnames(X)[ii]))
              next
            bb <- 1 - (rep(1, nrow(X)) %*% (X[, ii] * X[,
              iis]) == 0)
            if (jj != 0 & sum(bb) >= length(bb) - sum(yna))
              next
            mp = swp.f(coef[iis][bb == 1])
            coef[ii] <- mp + coef[ii]
            coef[iis] <- coef[iis] - mp * bb
        }
        coef
    }
    plvls <- permutes(as.numeric(names(table(lvls2[iiis]))))
    if (length(plvls) == 1)
        return(ipivot(X, coe, lvls2 == plvls, iis))
    sumcoe = 0
    for (iii in 1:ncol(plvls)) {
        coe.p = coe
        for (iiii in 1:nrow(plvls)) coe.p = ipivot(X, coe.p,
            lvls2 == plvls[iiii, iii], iis)
```

```
            sumcoe = sumcoe + coe.p
        }
        sumcoe/ncol(plvls)
    }
    co.p = co - co
    while (!isTRUE(all.equal(co.p, co))) {
        co.p = co
        for (kk in max(lvls2):1) {
            jj <- lvls1[lvls2 == kk][1] - 1
            for (jjj in jj:0) co <- pivot(X, co, lvls1 == jjj,
                lvls2 == kk)
        }
    }
    if (identical(swp.f, NE.median))
        co[1] = co[1] + med.y
    names(co) <- colnames(X)
    names(co)[(length(names(co)) - ly + 1):length(names(co))] = paste("res",
        1:ly)
    k = list()
    for (j in 0:max(lvls2)) k[[j + 1]] = co[lvls2 == j]
    x = rep(NA, length(yna))
    x[!yna] = k[[max(lvls2) + 1]]
    names(x) = paste("res", 1:ly)
    k[[max(lvls2) + 1]] = x
    co = unlist(k)
    cotab <- list.table3(round(co, 2), cbind(lvls1, lvls2), aa[[3]],
        dt)
    llist = length(cotab)
    dof = unlist(lapply(2:(llist - 2), function(ii) prod(dim(as.array(
        cotab[[ii]])) - 1)))
    dof = c(dof, dof.res = sum(!is.na(y)) - 1 - sum(dof))
    names(dof) <- c(attr(terms(mdl), "term.labels"), "residuals")
    obs = unlist(lapply(2:(llist - 2), function(ii) prod(dim(as.array(
```

```
        cotab[[ii]])))))
    obs = c(ly, obs)
    ll = list(vect = co, tbl = cotab, lvls = cbind(lvls1, lvls2),
        X = X, nterms = aa[[3]], dof = dof, obs = obs, mdl = mdl,
        dt = dt)
    class(ll) = c("lstpolish", "decomp")
    ll
}
```

D.2 incidence.matrix

```
function (mdl, dt = NULL)
{
    options(contrasts = c("contr.treatment", "contr.poly"))
    ddtt = ""
    if (!is.null(dt))
        ddtt = "dt$"
    tt <- attr(terms(mdl, data = dt), "term.labels")
    oo <- attr(terms(mdl, data = dt), "order")
    ff = ee = dd = c()
    for (ii in 1:length(tt)) {
        if (!is.null(dt))
            aa <- eval(parse(text = paste("model.matrix(~", tt[ii],
                ",data=dt)")))
        else aa <- eval(parse(text = paste("model.matrix(~",
            tt[ii], ")")))
        bb <- aa[, 1] - apply(as.matrix(aa[, -1]), 1, sum)
        if (sum(bb) == 0)
            aa <- aa[, -1]
        else {
            aa[, 1] <- bb
            dimnames(aa)[[2]][1] = paste(tt[ii], eval(parse(text = paste(
                "levels(", ddtt, tt[ii], ")")))[1], sep = "")
```

```
        }
        dd = cbind(dd, aa)
        ee = c(ee, rep(oo[ii], dim(aa)[2]))
        ff = c(ff, rep(ii, dim(aa)[2]))
    }
    ll = list(X = cbind(Intercept = 1, dd), lvls = cbind(lvls1 = c(0,
        ee), lvls2 = c(0, ff)), nterms = c("Intercept", tt))
    class(ll) = "lstX"
    ll
}
```

D.3 is.child

```
function (big, ss)
{
    if (big[1] == "" | ss == "Intercept")
        return(1)
    big = paste(big, collapse = ":")
    big = paste(":", big, ":", sep = "")
    ss = strsplit(ss, ":")
    ss = unlist(lapply(ss, function(xx) paste(":", xx, ":", sep = "")))
    prod(unlist(lapply(ss, function(xx) strsplit(big, xx) !=
        big)))
}
```

D.4 permutes

```
function (x)
{
    n = length(x)
    if (n == 1)
        return(x)
    tmp = sapply(1:n, function(i) rbind(x[i], permutes(x[-i])))
```

```
    matrix(tmp, nrow = n)
}
```

D.5 list.table3

```
function (co, lvls, nterms, dt)
{
    aal = list()
    for (j in 0:max(lvls[, 2])) aal[[j + 1]] = co[lvls[, 2] ==
        j]
    k <- lapply(1:(length(aal)), function(jj) {
        if (jj == length(aal))
            return(aal[[length(aal)]])
        kkk = c()
        for (ii in 1:length(aal[[jj]])) kkk <- cbind(kkk, unlist(strsplit(
            names(aal[[jj]])[ii], "\\:")))
        tapply(aal[[jj]], lapply(1:nrow(kkk), function(i) kkk[i,
            ]), paste)
    })
    names(k) = c(nterms, "residuals")
    idt = ""
    if (!is.null(dt))
        idt = "dt$"
    nldt = nterms[seq(2, leng = length(table(lvls[, 2][lvls[,
        1] == 1])))]
    ldt = eval(parse(text = paste("list(", paste(idt, nldt, collapse = ","),
        ")")))
    ldt = lapply(seq(along = ldt), function(ii) paste(nldt[ii],
        ldt[[ii]], sep = ""))
    k$res.tab = tapply(k$res, ldt, paste, collapse = ",")
    k
}
```

D.6 print.lstpolish

```
function (ll, ...)
{
    print(ll$tbl, ...)
}
```

D.7 print.lstX

```
function (ll, ...)
{
    print(ll$X, ...)
}
```

Appendix E

R functions in robande library

Any called function which is not presented here, can be found in previous appendices.

E.1 anova.decomp

```
function (tabdecomp, sclf = "lomedian", cf = 1.5, wins = 0.5)
{
    tab4.1 = tabdecomp
    tab3 <- decomp(mdl = tab4.1$mdl, dt = tab4.1$dt)
    tab4.2 = 0
    for (ii in 1:max(tab4.1$l[, 2])) tab4.2 = c(tab4.2, STm2(tab4.1$v[tab4.1$l[,
        2] == ii], ddff = tab4.1$dof[ii], cutoff = cf, scalf = sclf)$result)
    tab5 = tab4.1$v[1]
    for (ii in 1:max(tab4.1$l[, 2])) tab5 = c(tab5, n5gg.3(tab4.1$v[tab4.1$l[,
        2] == ii], tab4.2[tab4.1$l[, 2] == ii], wins))
    tab6 <- decomp(tab4.1$mdl, dt = tab4.1$dt, ico = tab5)
    tab7 <- tab4.2 * tab4.1$v - tab5 + tab6$v
    options(contrasts = c("contr.sum", "contr.sum"))
    kk <- t(t(tab4.1$X) * tab3$v)
    kky <- apply(kk, 1, sum)
    mdl.tmp = as.formula(paste("kky~", strsplit(as.character(tab4.1$mdl),
        "~")[3]))
    if (!is.null(tab4.1$dt))
```

```
    data.tmp = cbind(tab4.1$dt, kky)
else data.tmp = NULL
aa = anova(lm(mdl.tmp, data.tmp))[, 2]
kk <- t(t(tab4.1$X) * tab5)
kky <- apply(kk, 1, sum)
mdl.tmp = as.formula(paste("kky~", strsplit(as.character(tab4.1$mdl),
    "~")[3]))
if (!is.null(tab4.1$dt))
    data.tmp = cbind(tab4.1$dt, kky)
else data.tmp = NULL
bb = anova(lm(mdl.tmp, data.tmp))[, 2]
tagged = c()
for (ii in 1:max(tab4.1$l[, 2])) tagged = c(tagged, paste(names(
    tab4.1$v[tab4.2 == 1 & tab4.1$l[, 2] == ii]), collapse = ","))
Robust.ANOVA = cbind(DF = tab4.1$dof, Standard.MS = aa/tab4.1$dof,
    Inner.MS = bb/tab4.1$dof, MS.Changes.Percent = (aa -
        bb)/aa * 100)
rownames(Robust.ANOVA) = paste(rownames(Robust.ANOVA), tagged,
    sep = "->")
names(tab4.2) = names(tab5)
tmp = tab4.2
tmp[tmp == 0] = ""
tmp[tmp == "1"] = "*"
tmp = paste(tmp, round(tab4.1$v, 2))
names(tmp) = names(tab4.1$v)
pres = list.table3(tmp, tab4.1$l, tab4.1$n, tab4.1$dt)
robana = list(mean.dcmp = tab3$tbl, swp.dcmp = tab4.1$tbl,
    outliers = list.table3(tab4.2, tab4.1$l, tab4.1$n, tab4.1$dt),
    half.wins = tab5, inner = tab6$tbl, add.dcmp = tab7,
    Robust.ANOVA = list(formula = tab4.1$mdl, Robust.ANOVA = Robust.ANOVA),
    dt = tab4.1$dt, lvls = tab4.1$lvls, nterms = tab4.1$nterms,
    tab4.2, presentation = pres)
class(robana) = "lstrob"
```

```
    robana
}
```

E.2 n5gg.3

```
function (mm, mmtag, wins = 0.5)
{
    eee3 <- mm
    eee2 <- mmtag
    tmp1 = summary(eee3[eee2 != 1])[c(1, 6)] * wins
    eee2 = eee2 * sign(eee3)
    eee3[eee2 == -1] = tmp1[1]
    eee3[eee2 == 1] = tmp1[2]
    eee3
}
```

E.3 print.lstrob

```
function (robana, ...)
{
    print(robana$Robust.ANOVA, ...)
}
```

Appendix F

R functions to do the traditional decomposition

Any called function which is not presented here, can be found in previous appendices.

F.1 decom

```
function (mdl = 0, swp = "fibian", gf = "mean", eps = 0.01, rep = 100,
    dir = "first", cf = 1.5, scalf = "lomedian")
{
    bb = n3gg.2(mdl, gf)
    tab3 = n0gg(n2gg(bb[[2]]), "mean")
    if (swp == "NE.median") {
        med.bb = median(bb[[2]])
        bb[[2]] = bb[[2]] - med.bb
    }
    aa = n2gg(bb[[2]])
    if (dir == "avg")
        aa = n0gg.2(aa, swp)
    else aa = n0gg(aa, swp, st = dir)
    ptrn = n4gg(dim(aa), mdl, bb[[1]][1])
    if (swp == "NE.median")
        eval(parse(text = paste("aa", ptrn$index[1], "=med.bb+aa",
```

```
            ptrn$index[1])))
    tab4.1 = aa
    tab4.2tab5 = n5gg(aa, ptrn$i, 1, cf = cf, scalf = scalf)
    tab4.2 = tab4.2tab5$tag
    tab5 = tab4.2tab5$rep
    tab6 = n0gg(tab4.2tab5$rep, "mean")
    tab7 = tab4.2 * (tab4.1 - tab5) + tab6
    tab8a = cbind(n6gg(tab3, ptrn), n6gg(tab6, ptrn), n7gg(tab4.2,
        ptrn))
    tab8 = cbind(tab8a[, c(1, 2, 4, 5)], 1 - as.numeric(tab8a[,
        4])/as.numeric(tab8a[, 2]))
    colnames(tab8) = c("DF", "Std MS", "Inner MS", "# of exotics",
        "% of MS changes")
    tab10 = n10gg(mdl, as.numeric(tab8[, 2]), as.numeric(tab8[, 1]))
    tab11 = n10gg(mdl, as.numeric(tab8[, 3]), as.numeric(tab8[, 1]))
    ll = list(mean.decomposition = n8gg(tab3, ptrn), swp.decomposition =
        n8gg(tab4.1, ptrn), i.swp.decomposition = n8gg(tab4.2, ptrn),
        half_winsorized = n8gg(tab5, ptrn), mean.re_decomposition =
        n8gg(tab6, ptrn), additive.decomposition = n8gg(tab7, ptrn),
        Robust.ANOVA = tab8, tab10 = tab10, tab11 = tab11, ptrn = ptrn)
    class(ll) = c("lstoldpolish", "decom")
    ll
}
```

F.2 n3gg.2

```
function (mdl, gf)
{
    ttmm = as.character(attr(terms(mdl), "variables"))
    ccll = ""
    for (ii in 3:length(ttmm)) ccll <- paste(ccll, ttmm[ii],
        sep = ",")
    list(eval(parse(text = paste("tapply(", ttmm[2], ",", "list(",
```

```
        substring(ccll, 2), "),length)"))), eval(parse(text = paste("tapply(",
        ttmm[2], ",", "list(", substring(ccll, 2), "),", gf,
        ")"))))
}
```

F.3 n0gg.2

```
function (aa, swp, eps = 1e-14, rep = 100)
{
    for (l in 1:rep) {
        bb = aa
        per = permutes(1:length(dim(aa)))
        cc = array(0, dim(aa))
        for (j in 1:ncol(per)) {
            tmp = aa
            for (i in per[, j]) tmp <- n1gg(tmp, i, swp)
            cc <- tmp + cc
        }
        aa = cc/ncol(per)
        if (isTRUE(all.equal(bb, aa)))
            break
    }
    aa
}
```

F.4 n4gg

```
function (dimaa, ooo, repll)
{
    ff <- attr(terms(ooo), "factors")[-1, ]
    ff = cbind(rep(0, dim(ff)[1]), ff)
    ff <- (ff * (-2) + 1) * dimaa
    eee = c()
```

```
    for (i in 1:dim(ff)[2]) {
        cc = ff[, i]
        ee = "["
        for (j in 1:length(dimaa)) {
            ee = paste(ee, cc[j], ",")
        }
        ee = substr(ee, 1, nchar(ee) - 1)
        ee = paste(ee, "]")
        eee = c(eee, ee)
    }
    list(names = c("Overall", attr(terms(ooo), "term.labels")),
        index = eee, dof = apply(ff, 2, function(xx) {
            prod(xx[xx < 0] * (-1) - 2)
        }), n = apply(ff, 2, function(xx) {
            prod(xx[xx > 0] - 1)
        }) * repll, levels = c(0, attr(terms(ooo), "order")))
}
```

F.5 n5gg

```
function (aa, eee, l, cf = 1.5, scalf = "lomedian")
{
    aatag = aarep = aa
    eval(parse(text = paste("aatag", eee[1], "=0")))
    for (i in 2:length(eee)) {
        eee3 = eee1 = eval(parse(text = paste("aa", eee[i])))
        eee2 = STm2(as.array(eee1), scalf = scalf, cutoff = cf,
            ddff = prod(dim(as.array(eee1)) - 1))$r
        eval(parse(text = paste("aatag", eee[i], "=eee2")))
        tmp1 = summary(eee3[eee2 != 1])[c(1, 6)]/2
        eee2 = eee2 * sign(eee3)
        eee3[eee2 == -1] = tmp1[1]
        eee3[eee2 == 1] = tmp1[2]
```

```
        eval(parse(text = paste("aarep", eee[i], "=trunc(eee3)")))
    }
    list(tag = aatag, replaced = aarep)
}
```

F.6 n6gg

```
function (aa, ptrn)
{
    tmp = c()
    for (ii in 2:length(ptrn$n)) tmp = rbind(tmp, c(ptrn$dof[ii],
        eval(parse(text = paste("sum(aa", ptrn$index[ii], "^2)*ptrn$n[ii]/
        ptrn$dof[ii]")))))
    rownames(tmp) = ptrn$name[2:length(ptrn$n)]
    tmp
}
```

F.7 n7gg

```
function (aa, ptrn)
{
    cc = c()
    for (i in 2:length(ptrn$names)) cc = c(cc, sum(eval(parse(text =
        paste("aa", ptrn$index[i], "==1")))))
    cc
}
```

F.8 n10gg

```
function (mo, ms, df)
{
    for (kk in 1:(max(as.numeric(names(table(attr(terms(mo),
        "order"))))) - 1)) {
```

```
df1 <- df[attr(terms(mo), "order") == kk]
df2 <- df[attr(terms(mo), "order") == (kk + 1)]
ms1 <- ms[attr(terms(mo), "order") == kk]
ms2 <- ms[attr(terms(mo), "order") == (kk + 1)]
l1 = length(ms1)
l2 = length(ms2)
msm = matrix(0, l1, l2)
for (ii in 1:l1) for (jj in 1:l2) msm[ii, jj] <- log2(ms1[ii]) -
    log2(ms2[jj])
a <- attr(terms(mo), "factor")[, attr(terms(mo), "order") ==
    kk]
b <- attr(terms(mo), "factor")[, attr(terms(mo), "order") ==
    (kk + 1)]
ab = matrix(0, l1, l2)
for (ii in 1:l1) for (jj in 1:l2) {
    ab[ii, jj] <- sum(a[, ii] * as.matrix(b)[, jj]) == kk
}
colnames(msm) = attr(terms(mo), "term.labels")[attr(terms(mo),
    "order") == (kk + 1)]
rownames(msm) = attr(terms(mo), "term.labels")[attr(terms(mo),
    "order") == kk]
msm[ab == 0] = NA
msm[msm > 1] = NA
msm = apply(msm, 2, function(x) {
    x == apply(msm, 1, min, na.rm = NA)
})
msm[msm == FALSE] = NA
ms[attr(terms(mo), "order") == (kk + 1)] <- apply(rbind(msm *
    ms1, ms2) * rbind(msm * df1, df2), 2, sum, na.rm = TRUE)/
    apply(rbind(msm * df1, df2), 2, sum, na.rm = TRUE)
ms[attr(terms(mo), "order") == kk] <- (apply(msm, 1,
    sum, na.rm = TRUE) == 0) * ms1
df[attr(terms(mo), "order") == (kk + 1)] <- apply(rbind(msm *
```

```
            df1, df2), 2, sum, na.rm = TRUE)
        df[attr(terms(mo), "order") == kk] <- (apply(msm, 1,
            sum, na.rm = TRUE) == 0) * df1
    }
    cbind(attr(terms(mo), "term.labels"), ms, df)
    tmp = cbind(ms, df)
    rownames(tmp) = attr(terms(mo), "term.labels")
    tmp[tmp[, 2] != 0, ]
}
```

F.9 print.lstoldpolish

```
function (ll, ...)
{
    print(ll$swp.decomposition, ...)
}
```